Hidden Voice Commands

Nicholas Carlini
University of California, Berkeley

Pratyush Mishra
University of California, Berkeley

Tavish Vaidya
Georgetown University

Yuankai Zhang
Georgetown University

Micah Sherr
Georgetown University

Clay Shields
Georgetown University

David Wagner
University of California, Berkeley

Wenchao Zhou
Georgetown University

Abstract

Voice interfaces are becoming more ubiquitous and are now the primary input method for many devices. We explore in this paper how they can be attacked with *hidden voice commands* that are unintelligible to human listeners but which are interpreted as commands by devices.

We evaluate these attacks under two different threat models. In the black-box model, an attacker uses the speech recognition system as an opaque oracle. We show that the adversary can produce difficult to understand commands that are effective against existing systems in the black-box model. Under the white-box model, the attacker has full knowledge of the internals of the speech recognition system and uses it to create attack commands that we demonstrate through user testing are not understandable by humans.

We then evaluate several defenses, including notifying the user when a voice command is accepted; a verbal challenge-response protocol; and a machine learning approach that can detect our attacks with 99.8% accuracy.

1 Introduction

Voice interfaces to computer systems are becoming ubiquitous, driven in part by their ease of use and in part by decreases in the size of modern mobile and wearable devices that make physical interaction difficult. Many devices have adopted an always-on model in which they continuously listen for possible voice input. While voice interfaces allow for increased accessibility and potentially easier human-computer interaction, they are at the same time susceptible to attacks: Voice is a broadcast channel open to any attacker that is able to create sound within the vicinity of a device. This introduces an opportunity for attackers to try to issue unauthorized voice commands to these devices.

An attacker may issue voice commands to any device that is within speaker range. However, naïve attacks will be conspicuous: a device owner who overhears such a command may recognize it as an unwanted command and cancel it, or otherwise take action. This motivates the question we study in this paper: can an attacker create *hidden voice commands*, i.e., commands that will be executed by the device but which won't be understood (or perhaps even noticed) by the human user?

The severity of a hidden voice command depends upon what commands the targeted device will accept. Depending upon the device, attacks could lead to information leakage (e.g., posting the user's location on Twitter), cause denial of service (e.g., activating airplane mode), or serve as a stepping stone for further attacks (e.g., opening a web page hosting drive-by malware). Hidden voice commands may also be broadcast from a loudspeaker at an event or embedded in a trending YouTube video, compounding the reach of a single attack.

Vaidya et al. [40] showed that hidden voice commands are possible—attackers can generate commands that are recognized by mobile devices but are considered as noise by humans. Building on their work, we show more powerful attacks and then introduce and analyze a number of candidate defenses.

The contributions of this paper include the following:

- We show that hidden voice commands can be constructed even with very little knowledge about the speech recognition system. We provide a general attack procedure for generating commands that are likely to work with any modern voice recognition system. We show that our attacks work against Google Now's speech recognition system and that they improve significantly on previous work [40].
- We show that adversaries with significant knowledge of the speech recognition system can construct hidden voice commands that humans cannot understand at all.
- Finally, we propose, analyze, and evaluate a suite of detection and mitigation strategies that limit the effects of the above attacks.

Audio files for the hidden voice commands described in this paper are available for download at https://sites.google.com/site/hiddenvoicecommands16/ (hosted anonymously for double-blind review).

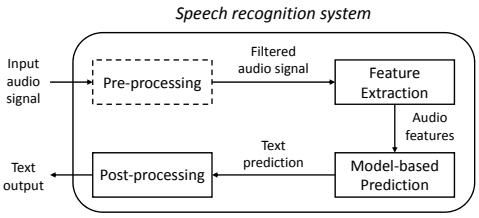

Figure 1: Overview of a typical speech recognition system.

2 Background and Related Work

To set the stage for the attacks that we present in §3 and §4, we briefly review how speech recognition works.

Figure 1 presents a high-level overview of a typical speech recognition procedure, which consists of the following four steps: pre-processing, feature extraction, model-based prediction, and post-processing. Preprocessing performs initial speech/non-speech identification by filtering out frequencies that are beyond the range of a human voice and eliminating time periods where the signal energy falls below a particular threshold. This step only does rudimentary filtering, but still allows non-speech signals to pass through the filter if they pass the energy-level and frequency checks.

The second step, feature extraction, splits the filtered audio signal into short (usually around 20 ms) frames and extracts features from each frame. The feature extraction algorithm used in speech recognition is almost always the Mel-frequency cepstral (MFC) transform [20, 41]. We describe the MFC transform in detail in Appendix A, but at a high level it can be thought of as a transformation that extracts the dominant frequencies from the input.

The model-based prediction step takes as input the extracted features, and matches them against an existing model built offline to generate text predictions. The technique used in this step can vary widely: some systems use Hidden Markov Models, while many recent systems have begun to use recurrent neural networks (RNNs).

Finally, a post-processing step ranks the text predictions by employing additional sources of information, such as grammar rules or locality of words.

Related work. Unauthorized voice commands have been studied by Diao et al. [12] and Jang et al. [21] who demonstrate that malicious apps can inject synthetic audio or play commands to control smartphones. Unlike in this paper, these attacks use non-hidden channels that are understandable by a human listener.

Similar to our work, Kasmi and Lopes Esteves [23] consider the problem of *covert* audio commands. There, the authors inject voice commands by transmitting FM signals that are received by a headset. In our work, we do not require the device to have an FM antenna (which is not often present) and we obfuscate the voice command so that it is not human-recognizable. Schlegel et al. [35] show that malicious apps can eavesdrop and record phone calls to extract sensitive information. Our work differs in that it exploits targeted devices' existing functionality (i.e., speech recognition) and does not require the installation of malicious apps.

Earlier work by Vaidya et al. [40] introduces obfuscated voice commands that are accepted by voice interfaces. Our work significantly extends their black-box approach by (i) evaluating the effectiveness of their attacks under realistic scenarios, (ii) introducing more effective "white-box" attacks that leverage knowledge of the speech recognition system to produce machine-understandable speech that is almost never recognized by humans, (iii) formalizing the method of creating hidden voice commands, and (iv) proposing and evaluating defenses.

Image recognition systems have been shown to be vulnerable to attacks where slight modifications to only a few pixels can change the resulting classification dramatically [17, 19, 25, 37]. Our work has two key differences. First, feature extraction for speech recognition is significantly more complex than for images; this is one of the main hurdles for our work. Second, attacks on image recognition have focused on the case where the adversary is allowed to directly modify the electronic image. In contrast, our attacks work "over the air"; that is, we create audio that when played and recorded is recognized as speech. The analogous attack on image recognition systems would be to create a physical object which appears benign, but when photographed, is classified incorrectly. As far as we know, no one has demonstrated such an attack on image recognition systems.

More generally, our attacks can be framed as an evasion attack against machine learning classifiers: if f is a classifier and A is a set of acceptable inputs, given a desired class y, the goal is to find an input $x \in A$ such that $f(x) = y$. In our context, f is the speech recognition system, A is a set of audio inputs that a human would not recognize as speech, and y is the text of the desired command. Attacks on machine learning have been studied extensively in other contexts [1, 4, 5, 10, 13, 22, 31, 39]; In particular, Fawzi et al. [14] develop a rigorous framework to analyze the vulnerability of various types of classifiers to adversarial perturbation of inputs. They demonstrate that a minimal set of adversarial changes to input data is enough to fool most classifiers into misclassifying the input. Our work is different in two key respects: (i) the above caveats for image recognition systems still apply, and moreover, (ii) their work does not necessarily aim to create inputs that are misclassified into a particular category; but rather that it is just misclassified. On the other hand, we aim to craft inputs that are recognized as potentially sensitive commands.

Finally, Fredrikson et al. [15] attempt to invert machine learning models to learn private and potentially sensitive data in the training corpus. They formulate their task as an optimization problem, similar to our white-box approach, but they (i) test their approach primarily on image recognition models, which, as noted above, are easier to fool, and (ii) do not aim to generate adversarial inputs, but rather only extract information about individual data points.

3 Black-box Attacks

We first show that under a weak set of assumptions an attacker with no internal knowledge of a voice recognition system can generate hidden voice commands that are difficult for human listeners to understand. We refer to these as *obfuscated commands*, in contrast to unmodified and understandable *normal commands*.

These attacks were first proposed by Vaidya et al. [40]. This section improves upon the efficacy and practicality of their attacks and analysis by (i) carrying out and testing the performance of the attacks under more practical settings, (ii) considering the effects of background noise, and (iii) running the experiments against Google's improved speech recognition service [33].

3.1 Threat model & attacker assumptions

In this black-box model the adversary does not know the specific algorithms used by the speech recognition system. We assume that the system extracts acoustic information through some transform function such as an MFC, perhaps after performing some pre-processing such as identifying segments containing human speech or removing noise. MFCs are commonly used in current-generation speech recognition systems [20, 41], making our results widely applicable, but not limited to such systems.

We treat the speech recognition system as an oracle to which the adversary can pose transcription tasks. The adversary can thus learn how a particular obfuscated audio signal is interpreted. We do not assume that a particular transcription is guaranteed to be consistent in the future. This allows us to consider speech recognition systems that apply randomized algorithms as well as to account for transient effects such as background noise and environmental interference.

Conceptually, this model allows the adversary to iteratively develop obfuscated commands that are increasingly difficult for humans to recognize while ensuring, with some probability, that they will be correctly interpreted by a machine. This trial-and-error process occurs in advance of any attack and is invisible to the victim.

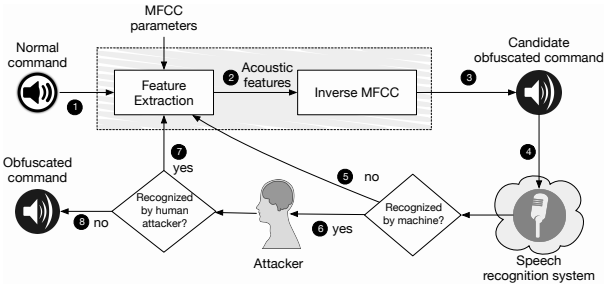

Figure 2: Adversary's workflow for producing an obfuscated audio command from a normal command.

3.2 Overview of approach

We rerun the black-box attack proposed by Vaidya et al. [40] as shown in Figure 2. The attacker's goal is to produce an obfuscated command that is accepted by the victim's speech recognition system but is indecipherable by a human listener.

The attacker first produces a normal command that it wants executed on the targeted device. To thwart individual recognition the attacker may use a text-to-speech engine, which we found is generally correctly transcribed. This command is then provided as input (Figure 2, step ❶) to an *audio mangler*, shown as the grey box in the figure. The audio mangler performs an MFC with a starting set of parameters on the input audio, and then performs an inverse MFC (step ❷) that additionally adds noise to the output. By performing the MFC and then inverting the obtained acoustic features back into an audio sample, the attacker is in essence attempting to remove all audio features that are not used in the speech recognition system but which a human listener might use for comprehension.

Since the attacker does not know the MFC features used by the speech recognition system, experimentation is required. First, the attacker provides the *candidate obfuscated audio* that results from the MFC→inverse-MFC process (step ❸) to the speech recognition system (step ❹). If the command is not recognized then the attacker must update the MFC parameters to ensure that the result of the MFC→inverse-MFC transformation will yield higher fidelity audio (step ❺).

If the candidate obfuscated audio is interpreted correctly (step ❻), then the human attacker tests if it is human understandable. This step is clearly subjective and, worse, is subject to *priming* effects [28] since the attacker already knows the correct transcription. The attacker may solicit outside opinions by crowdsourcing. If the obfuscated audio is too easily understood by humans the attacker discards the candidate and generates new candidates by adjusting the MFC parameters to produce lower fidelity audio (step ❼). Otherwise, the can-

Table 1: MFC parameters tuned to produce obfuscated audio.

Parameter	Description
`wintime`	time for which the signal is considered constant
`hoptime`	time step between adjacent windows
`numcep`	number of cepstral coefficients
`nbands`	no. of warped spectral bands for aggregating energy levels

didate obfuscated audio command—which is recognized by machines but not by humans—is used to conduct the actual attack (step ❽).

3.3 Experimental setup

We obtained the audio mangling program used by Vaidya et al. [40]. Conforming to their approach, we also manually tune four MFC parameters to mangle and test audio using the workflow described in §3.2 to determine the ranges for human and machine perception of voice commands. The list of modified MFC parameters is presented in Table 1.

Our voice commands consisted of the phrases "OK google", "call 911", and "turn on airplane mode". These commands were chosen to represent a variety of potential attacks against personal digital assistants. Voice commands were played using Harmon Kardon speakers, model number HK695–01,13, in a conference room measuring approximately 12 by 6 meters, 2.5 meters tall. Speakers were on a table approximately three meters from the phones. The room contained office furniture and projection equipment. We measured a background noise level ($P_{\text{dB}}^{\text{noise}}$) of approximately 53 dB.

We tested the commands against two smart phones, a Samsung Galaxy S4 running Android 4.4.2 and Apple iPhone 6 running iOS 9.1 with Google Now app version 9.0.60246. Google's recently updated [33] default speech recognition system was used to interpret the commands. In the absence of injected ambient background noise, our sound level meter positioned next to the smartphones measured the median intensity of the voice commands to be approximately 88 dB.

We also projected various background noise samples collected from SoundBible [9], recorded from a casino, classroom, shopping mall, and an event during which applause occurred. We varied the volume of these background noises—thus artificially adjusting the signal-to-noise ratio—and played them through eight overhead JBL in-ceiling speakers. We placed a Kinobo "Akiro" table mic next to our test devices and recorded all audio commands that we played to the devices for use in later experiments, described below.

3.4 Evaluation

Attack range. We found that the phone's speech recognition system failed to identify speech when the speaker was located more than 3.5 meters away or when the perceived SNR was less than 5 dB. We conjecture that the speech recognition system is designed to discard far away noises, and that sound attenuation further limits the attacker's possible range. While the attacker's locality is clearly a limitation of this approach, there are many attack vectors that allow the attacker to launch attacks within a few meters of the targeted device, such as obfuscated audio commands embedded in streaming videos, overhead speakers in offices, elevators, or other enclosed spaces, and propagation from other nearby phones.

Machine understanding. Table 2 shows a side-by-side comparison of human and machine understanding, for both normal and obfuscated commands.

The "machine" columns indicate the percentage of trials in which a command is correctly interpreted by the phone, averaged over the various background noises. Here, our sound meter measured the signal's median audio level at 88 dB and the background noise at 73 dB, corresponding to a signal-to-noise ratio of 15 dB.

Across all three commands, the phones correctly interpreted the normal versions 85% of the time. This accuracy decreased to 60% for obfuscated commands.

We also evaluate how the amplitude of background noise affects machine understanding of the commands. Figure 3 shows the percentage of voice commands that are correctly interpreted by the phones ("success rate") as a function of the SNR (in dB) using the Mall background noise. Note that a higher SNR denotes more favorable conditions for speech recognition. Generally, Google's speech recognition engine correctly transcribes the voice commands and activates the phone. The accuracy is higher for normal commands than obfuscated commands, with accuracy improving as SNR increases. In all cases, the speech recognition system is able to perfectly understand and activate the phone functionality in at least some configurations—that is, all of our obfuscated audio commands work at least *some* of the time. With little background noise, the obfuscated commands work extremely well and are often correctly transcribed at least 80% of the time. Appendix B shows detailed results for additional background noises.

Human understanding. To test human understanding of the obfuscated voice commands, we conducted a study on Amazon Mechanical Turk[1], a service that pays

[1]**Note on ethics:** Before conducting our Amazon Mechanical Turk experiments, we submitted an online application to our institution's IRB. The IRB responded by stating that we were exempt from IRB. Irrespective of our IRB, we believe our experiments fall well within the

Table 2: Black-box attack results. The "machine" columns report the percentage of commands that were correctly interpreted by the tested smartphones. The percentage of commands that were correctly understood by humans (Amazon Turk workers) is shown under the "human" columns. For the latter, the authors assessed whether the Turk workers correctly understood the commands.

	Ok Google		**Turn on airplane mode**		**Call 911**	
	Machine	*Human*	*Machine*	*Human*	*Machine*	*Human*
Normal	90% (36/40)	89% (356/400)	75% (30/40)	69% (315/456)	90% (36/40)	87% (283/324)
Obfuscated	95% (38/40)	22% (86/376)	45% (18/40)	24% (109/444)	40% (16/40)	94% (246/260)

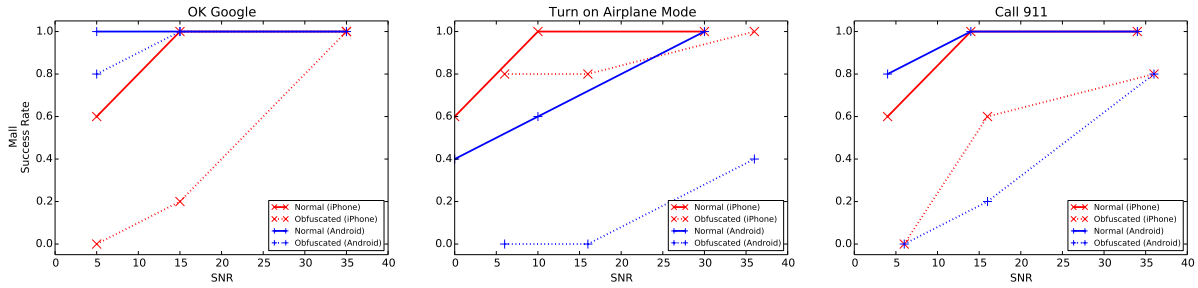

Figure 3: Machine understanding of normal and obfuscated variants of "OK Google", "Turn on Airplane Mode", and "Call 911" voice commands under Mall background noise. Each graph shows the measured average success rate (the fraction of correct transcripts) on the y-axis as a function of the signal-to-noise ratio.

human workers to complete online tasks called Human Intelligence Tasks (HITs). Each HIT asks a user to transcribe several audio samples, and presents the following instructions: "We are conducting an academic study that explores the limits of how well humans can understand obfuscated audio of human speech. The audio files for this task may have been algorithmically modified and may be difficult to understand. Please supply your best guess to what is being said in the recordings."

We constructed the online tasks to minimize priming effects—no worker was presented with both the normal and obfuscated variants of the same command. Due to this structuring, the number of completed tasks varies among the commands as reflected in Table 2 under the "human" columns.

We additionally required that workers be over 18 years of age, citizens of the United States, and non-employees of our institution. Mechanical Turk workers were paid $1.80 for completing a HIT, and awarded an additional $0.20 for each correct transcription. We could not prevent the workers from replaying the audio samples multiple times on their computers and the workers were incentivized to do so, thus our results could be considered conservative: if the attacks were mounted in practice, device owners might only be able to hear an attack once.

basic principles of ethical research. With respect in particular to beneficence, the Mechanical Turk workers benefited from their involvement (by being compensated). The costs/risks were extremely low: workers were fully informed of their task and no subterfuge occurred. No personal information—either personally identifiable or otherwise—was collected and the audio samples consisted solely of innocuous speech that is very unlikely to offend (e.g., commands such as "OK Google").

To assess how well the Turk workers understood normal and obfuscated commands, four of the authors compared the workers' transcriptions to the correct transcriptions (e.g., "OK Google") and evaluated *whether both had the same meaning*. Our goal was not to assess whether the workers correctly heard the obfuscated command, but more conservatively, whether their perception conformed with the command's meaning. For example, the transcript "activate airplane functionality" indicates a failed attack even though the transcription differs significantly from the baseline of "turn on airplane mode".

Values shown under the "human" columns in Table 2 indicate the fraction of total transcriptions for which the survey takers believed that the Turk worker understood the command. Each pair of authors had an agreement of over 95% in their responses, the discrepancies being mainly due to about 5% of responses in which one survey taker believed they matched but the others did not. The survey takers were presented only with the actual phrase and transcribed text, and were blind to whether or not the phrase was an obfuscated command or not.

Turk workers were fairly adept (although not perfect) at transcribing normal audio commands: across all commands, we assessed 81% of the Turkers' transcripts to convey the same meaning as the actual command.

The workers' ability to understand obfuscated audio was considerably less: only about 41% of obfuscated commands were labeled as having the same meaning as the actual command. An interesting result is that the black-box attack performed far better for some commands than others. For the "Ok Google" command, we

decreased human transcription accuracy fourfold without any loss in machine understanding.

"Call 911" shows an anomaly: human understanding increases for obfuscated commands. This is due to a tricky part of the black-box attack workflow: the attacker must manage priming effects when choosing an obfuscated command. In this case, we believed the "call 911" candidate command to be unintelligible; these results show we were wrong. A better approach would have been to repeat several rounds of crowdsourcing to identify a candidate that was not understandable; any attacker could do this. It is also possible that among our US reviewers, "call 911" is a common phrase and that they were primed to recognize it outside our study.

Objective measures of human understanding: The analysis above is based on the authors' assessment of Turk workers' transcripts. In Appendix C, we present a more objective analysis using the Levenshtein edit distance between the true transcript and the Turkers' transcripts, with phonemes as the underlying alphabet.

We posit that our (admittedly subjective) assessment is more conservative, as it directly addresses human *understanding* and considers attacks to fail if a human understands the meaning of a command; in contrast, comparing phonemes measures something slightly different—whether a human is able to reconstruct the *sounds* of an obfuscated command—and does not directly capture understanding. Regardless, the phoneme-based results from Appendix C largely agree with those presented above.

4 White-box Attacks

We next consider an attacker who has knowledge of the underlying voice recognition system. To demonstrate this attack, we construct hidden voice commands that are accepted by the open-source CMU Sphinx speech recognition system [24]. CMU Sphinx is used for speech recognition by a number of apps and platforms[2], making it likely that these whitebox attacks are also practical against these applications.

4.1 Overview of CMU Sphinx

CMU Sphinx uses the Mel-Frequency Cepstrum (MFC) transformation to reduce the audio input to a smaller dimensional space. It then uses a Gaussian Mixture Model (GMM) to compute the probabilities that any given piece of audio corresponds to a given phoneme. Finally, using a Hidden Markov Model (HMM), Sphinx converts the phoneme probabilities to words.

[2]Systems that use CMU Sphinx speech recognition include the Jasper open-source personal digital assistant and Gnome Desktop voice commands. The Sphinx Project maintains a list of software that uses Sphinx at http://cmusphinx.sourceforge.net/wiki/sphinxinaction.

The purpose of the MFC transformation is to take a high-dimensional input space—raw audio samples—and reduce its dimensionality to something which a machine learning algorithm can better handle. This is done in two steps. First, the audio is split into overlapping frames.

Once the audio has been split into frames, we run the MFC transformation on each frame. The Mel-Frequency Cepstrum Coefficients (MFCC) are the 13-dimensional values returned by the MFC transform.

After the MFC is computed, Sphinx performs two further steps. First, Sphinx maintains a running average of each of the 13 coordinates and subtracts off the mean from the current terms. This normalizes for effects such as changes in amplitude or shifts in pitch.

Second, Sphinx numerically estimates the first and second derivatives of this sequence to create a 39-dimensional vector containing the original 13-dimensional vector, the 13-dimensional first-derivative vector, and the 13-dimensional-second derivative vector.

Note on terminology: For ease of exposition and clarity, in the remainder of this section, we call the output of the MFCC function *13-vectors*, and refer to the output after taking derivatives as *39-vectors*.

The Hidden Markov Model. The Sphinx HMM acts on the sequence of 39-vectors from the MFCC. States in the HMM correspond to phonemes, and each 39-vector is assigned a probability of arising from a given phoneme by a Gaussian model, described next. The Sphinx HMM is, in practice, much more intricate: we give the complete description in Appendix A.

The Gaussian Mixture Model. Each HMM state yields some distribution on the 39-vectors that could be emitted while in that state. Sphinx uses a GMM to represent this distribution. The GMMs in Sphinx are a mixture of eight Gaussians, each over $\mathbb{R}^{39}$. Each Gaussian has a mean and standard deviation over every dimension. The probability of a 39-vector v is the sum of the probabilities from each of the 8 Gaussians, divided by 8. For most cases we can approximate the sum with a maximization, as the Gaussians typically have little overlap.

4.2 Threat model

We assume the attacker has complete knowledge of the algorithms used in the system and can interact with them at will while creating an attack. We also assume the attacker knows the parameters used in each algorithm.

We use knowledge of the coefficients for each Gaussian in the GMM, including the mean and standard deviation for each dimension and the importance of each Gaussian. We also use knowledge of the dictionary file in order to turn words into phonemes. An attacker could reconstruct this file without much effort.

4.3 Simple approach

Given this additional information, a first possible attack would be to use the additional information about exactly what the MFCC coefficients are to re-mount the the previous black-box attack.

Instead of using the MFCC inversion process described in §3.2, this time we implement it using gradient descent—a generic optimization approach for finding a good solution over a given space—an approach which can be generalized to arbitrary objective functions.

Gradient descent attempts to find the minimum (or maximum) value of an objective function over a multi-dimensional space by starting from an initial point and traveling in the direction which reduces the objective most quickly. Formally, given a smooth function f, gradient descent picks an initial point x_0 and then repeatedly improves on it by setting $x_{i+1} = x_i + \varepsilon \cdot \nabla f(x_0)$ (for some small ε) until we have a solution which is "good enough".

We define the objective function $f(x) = (\text{MFCC}(x) - y)^2 \cdot z$, where x is the input frame, y is the target MFCC vector, and z is the relative importance of each dimension. Setting $z = (1, 1, \ldots, 1)$ takes the L^2 norm as the objective.

Gradient descent is not guaranteed to find the global optimal value. For many problems it finds only a *local* optimum. Indeed, in our experiments we have found that gradient descent only finds local optima, but this turns out to be sufficient for our purposes.

We perform gradient descent search one frame at a time, working our way from the first frame to the last. For the first frame, we allow gradient descent to pick any 410 samples. For subsequent frames, we fix the first 250 samples as the last 250 of the preceding frame, and run gradient descent to find the best 160 samples for the rest of the frame.

As it turns out, when we implement this attack, our results are no better than the previous black-box-only attack. Below we describe our improvements to make attacks completely unrecognizable.

4.4 Improved attack

To construct hidden voice commands that are more difficult for humans to understand, we introduce two refinements. First, rather than targeting a specific sequence of MFCC vectors, we start with the target phrase we wish to produce, derive a sequence of phonemes and thus a sequence of HMM states, and attempt to find an input that matches that sequence of HMM states. This provides more freedom by allowing the attack to create an input that yields the same sequence of phonemes but generates a different sequence of MFCC vectors.

Second, to make the attacks difficult to understand, we use as few frames per phoneme as possible. In normal human speech, each phoneme might last for a dozen frames or so. We try to generate synthetic speech that uses only four frames per phoneme (a minimum of three is possible—one for each HMM state). The intuition is that the HMM is relatively insensitive to the number of times each HMM state is repeated, but humans are sensitive to it. If Sphinx does not recognize the phrase at the end of this process, we use more frames per phoneme.

For each target HMM state, we pick one Gaussian from that state's GMM. This gives us a sequence of target Gaussians, each with a mean and standard deviation.

Recall that the MFC transformation as we defined it returns a 13-dimensional vector. However, there is a second step which takes sequential derivatives of 13-vectors to produce 39-vectors. The second step of our attack is to pick these 13-vectors so that after we take the derivatives, we maximize the likelihood score the GMM assigns to the resulting 39-vector. Formally, we wish to find a sequence y_i of 39-dimensional vectors, and x_i of 13-dimensional vectors, satisfying the derivative relation

$$y_i = (x_i, x_{i+2} - x_{i-2}, (x_{i+3} - x_{i-1}) - (x_{i+1} - x_{i-3}))$$

and maximizing the likelihood score

$$\prod_i \exp\left\{ \sum_{j=1}^{39} \frac{\alpha_i^j - (y_i^j - \mu_i^j)^2}{\sigma_i^j} \right\}$$

where μ_i, σ_i, and α_i are the mean, standard deviation, and importance vectors respectively.

We can solve this problem exactly by using the least-squares method. We maximize the *log-likelihood*,

$$\log \prod_i \exp\left\{ \sum_j \frac{-\alpha_i^j + (y_i^j - \mu_i^j)^2}{\sigma_i^j} \right\} = \sum_i \sum_j \frac{-\alpha_i^j + (y_i^j - \mu_i^j)^2}{\sigma_i^j}$$

The log-likelihood is a sum of squares, so maximizing it is a least-squares problem: we have a linear relationship between the x and y values, and the error is a squared difference.

In practice we cannot solve the full least squares problem all at once. The Viterbi algorithm only keeps track of the 100 best paths for each prefix of the input, so if the global optimal path had a prefix that was the 101st most likely path, it would be discarded. Therefore, we work one frame at a time and use the least squares approach to find the next best frame.

This gives us three benefits: First, it ensures that at every point in time, the next frame is the best possible given what we have done so far. Second, it allows us to try all eight possible Gaussians in the GMM to pick the one which provides the highest score. Third, it makes our approach more resilient to failures of gradient descent.

Sometimes gradient descent cannot hit the 13-vector suggested by this method exactly. When this happens, the error score for subsequent frames is based on the actual 13-vector obtained by gradient descent.

Complete description. We first define two subroutines to help specify our attack more precisely. $\mathsf{LSTDERIV}(f, \bar{g}, g)$ accepts a sequence of 13-vectors f that have already been reached by previous iterations of search, the desired 39-vector sequence $\bar{g}$, and one new 39-vector g; it uses least squares to compute the next 13-vector which should be targeted along with the least-squares error score. We specify the algorithm in full detail in Appendix E.

$\mathsf{GRADDESC}(s,t)$ accepts the previous frame $s \in \mathbb{R}^{410}$ and a target 13-vector t, and returns a frame $\hat{s} \in \mathbb{R}^{410}$ such that $\hat{s}$ matches s in the 250 entries where they overlap and $\mathrm{MFCC}(\hat{s})$ is as close to t as possible. More precisely, it looks for a 160-dimensional vector x that minimizes $f(x) = ||\mathrm{MFCC}(s_{160\ldots410}||x) - s||_2$, where $||$ is concatenation, and returns $s_{160\ldots410}||x$. We use the Newton Conjugate-Gradient algorithm for gradient descent and compute the analytic derivative for efficiency.

Our full algorithm works as follows:

1. In the following, f will represent a sequence of chosen 13-vectors (initially empty), $\bar{g}$ a sequence of target 39-vectors, s the audio samples to return, and i the iteration number (initially 0).
2. Given the target phrase, pick HMM states h_i such that each state corresponds to a portion of a phoneme of a word in the phrase.
3. Let g_i^j be the 39-vector corresponding to the mean of the j^{th} Gaussian of the GMM for this HMM state. One of these will be the target vector we will try to invert.
4. For each j, solve the least squares problem $(s_j, d_j) = \mathsf{LSTDERIV}(f, \bar{g}, g_i^j)$ and set $\hat{j} = \arg\min_j s_j$ and $\bar{d} = d_{\hat{j}}$ to obtain a sequence of 13-vectors $\bar{d}_0$ to $\bar{d}_{i+6}$. Let $\bar{d}_i$ be the "target 13-vector" t. Append the 39-vector corresponding to t to $\bar{g}$.
5. Use gradient descent to get $\hat{s} = \mathsf{GRADDESC}(s,t)$. Let $s := \hat{s}$. Append $\mathrm{MFCC}(s)$ to f.
6. Repeat for the next i from step 3 until all states are completed.

4.5 Playing over the air

The previous attacks work well when we feed the audio file directly into Sphinx. However, Sphinx could not correctly transcribe recordings made by playing the audio using speakers. We developed three approaches to solve this complication:

Make the audio easier to play over speaker. Gradient descent often generates audio with very large spikes.

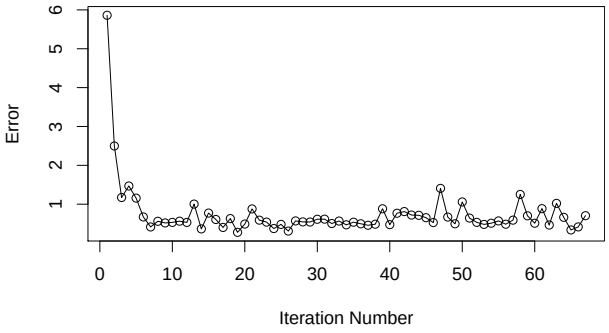

Figure 4: Incorporating actually playing the audio over the speakers into the gradient descent significantly reduces the error. The plot is of the L^2 norm of the error of from the target feature vector to the actually recorded feature vector, over time.

It's physically impossible for the speaker membrane to move quickly enough to accurately reproduce these spikes. We modified gradient descent to penalize waveforms that a speaker cannot reproduce. In particular, we add a penalty for large second derivatives in the signal, with the hope that gradient descent finds solutions that do not include such large spikes.

Predict the MFCC of played audio. Even with this penalty, the audio is still not perfectly playable over a speaker. When we compared the waveform of the played audio and recorded audio, they had significant differences. To address this, we built a model to predict the MFCC when a file is played through a speaker and recorded. Recall that the MFCC transformation essentially computes the function $C\log(B\,\|Ax\|^2)$.

By playing and recording many audio signals, we learned new matrices $\hat{A}$, $\hat{B}$, $\hat{C}$ so that for each played frame x and recorded frame y, $C\log(B\,\|Ay\|^2)$ is close to $\hat{C}\log(\hat{B}\,\|\hat{A}x\|^2)$. We computed $\hat{A}$, $\hat{B}$, $\hat{C}$ by solving a least-squares problem. This was still not enough for correct audio recognition, but it did point us in a promising direction.

Play the audio during gradient descent. The ideas above are not enough for recognition of recorded audio. To see what is going on here, we compare the MFCC of the played audio (after recording it) and the initial audio (before playing it). We found the correlation to be very high ($r = .97$ for the important coefficients).

Based on this observation, we augment our algorithm to include an outer iteration of gradient descent. Given a target MFCC we first run our previous gradient descent algorithm to find a sound sequence which (before playing over the speaker) reaches the target MFCC. Then, we play and record it over the speaker. We obtain from this the actual MFCC. We then adjust the target MFCC by the difference between what was received and what is desired.

We implemented this approach. Figure 4 plots the L^2 error (the difference between the target MFCC and what is actually recorded during each iteration of our algorithm) over time. By repeating this procedure 50 times and taking the frame with the minimum noise, we obtain an audio file that is correctly recognized by Sphinx after being played over the speaker.

Since we perform 50 iterations of the inner gradient descent per frame, and each iteration takes 30 seconds, our approach takes nearly 30 hours to find a valid attack sample. In practice, sometimes this process can take even longer; since we are recording audio, if the microphone picks up too much background noise, we must discard the recorded sample and try again. We have built in an error-detection system to mitigate these effects.

This might seem like a high cost to generate one attack sample. However, once generated, we can reuse the obfuscated audio on that speaker forever. Even though the setup cost is high, it must only be performed once; thereafter the same audio file can be used repeatedly.

4.6 Evaluation

Machine comprehension. For the former, we apply the above techniques and generate three audio commands: "okay google, take a picture", "okay google, text 12345", and "okay google, browse to evil.com". The speech recognition system is an instance of CMU Sphinx version 4-1.0beta6.

We determined the minimum number of frames per phoneme that is sufficient to allow Sphinx to recognize the command. Some words are more difficult to create correctly than others, and thus require more frames per phoneme. Detailed results can be found in Appendix F. When we modify the lengths of the phonemes to account for this data, *over 90% of generated phrases are correctly recognized by Sphinx*.

To evaluate our attack playing over a microphone, we equipped our computer with an external Blue Snowball microphone. Obfuscated audio commands are played from an Altec Lansing ASC45.1 speaker located 0.5 meters from the microphone. All of our evaluated phrases were commands recorded from the microphone. We trained our MFCC vector predictor with one minute (16 million samples) of audio played over this speaker in a large room to minimize echoes. The room was isolated, and thus we encountered nearly no background noise. The attack took 32 hours to generate the command for "okay google, take a picture", but required no human intervention, using the same number of frames chosen from the previous attack for a total of 101 frames.

Once the file was generated, we played the command over the speaker 100 times. Sphinx was able to recognize it in 82 of the instances. This is to be expected—for a given audio played multiple times, there is a very high (99%+) correlation between the recorded MFCC values.

Table 3: White-box attack results. Percentages show successful comprehension of Normal and Obfuscated version of voice commands.

	Command
Normal	74% (230/310)
Obfuscated	0% (1/377)

Human comprehension. As with our black-box attack, we evaluate our obfuscated commands with the same Amazon Mechanical Turk study. Workers were blind to how the obfuscated audio was generated. As a baseline, we asked workers to transcribe a phrase spoken by a human naturally. The results show that our white-box attack is significantly better than the black-box attack. No worker was able to transcribe more than half of the words in the phrase correctly, and 75% of workers transcriptions shared no words in common with the actual phrase.

Four of the authors independently evaluated 377 transcriptions of attack commands generated by our white-box attack and compared them with correct transcriptions to answer whether both transcriptions had the same meaning. The four authors agreed on over 95% of instances. Table 3 contains the results, showing that normal commands were correctly transcribed by workers 74% of the time, but obfuscated commands were almost never transcribed correctly. A comparison using phoneme-level edit distance yields similar results; see Appendix C.2.

While these results indicate that obfuscated commands generated using our white-box attack are very difficult to understand, we conducted a second study to determine if users actually thought the audio was human speech or just noise. Specifically, we created audio samples of a human speaking a phrase, followed by an obfuscated (different) phrase, and finally a human speaking a third different phrase. In this study we were interested in seeing if the worker would try to transcribe the obfuscated speech at all, or leave it out entirely.

Transcription accuracy was 80% for the first and last commands given by a human speaking. Only 24% of users attempted to transcribe the obfuscated speech. This study clearly demonstrates that when given a choice about what they viewed as speech and not-speech, the majority of workers believed our audio was not speech.

5 Defenses

We are unaware of any device or system that currently defends against obfuscated voice commands. In this section, we explore potential defenses for hidden voice com-

mands across three dimensions: *defenses that notify*, *defenses that challenge*, and *defenses that detect and prohibit*. The defenses described below are not intended to be exhaustive; they represent a first examination of potential defenses against this new threat.

5.1 Defenses that notify

As a first-line of defense we consider defenses that alert the user when the device interprets voice commands, though these will only be effective when the device operator is present and notification is useful (e.g., when it is possible to undo any performed action).

The "Beep", the "Buzz" and the "Lightshow". These defenses are very simple: when the device receives a voice command, it notifies the user, e.g., by beeping. The goal is to make the user aware a voice command was accepted. There are two main potential issues with "the Beep": (i) attackers may be able to mask the beep, or (ii) users may become accustomed to their device's beep and begin to ignore it. To mask the beep, the attacker might play a loud noise concurrent with the beep. This may not be physically possible depending on the attacker's speakers and may not be sufficiently stealthy depending on the environment as the noise require can be startling.

A more subtle attack technique is to attempt to mask the beep via noise cancellation. If the beep were a single-frequency sine wave an attacker might be able to cause the user to hear nothing by playing an identical frequency sine wave that is out of phase by exactly half a wavelength. We evaluated the efficacy of this attack by constructing a mathematical model that dramatically oversimplifies the attacker's job and shows that even this simplified "anti-beep" attack is nearly impossible. We present a more detailed evaluation of beep cancelation in Appendix D.

Some devices might inform the user when they interpret voice commands by vibrating ("the buzz") or by flashing LED indicators ("the lightshow"). These notifications also assume that the user will understand and heed such warnings and will not grow accustomed to them. To differentiate these alerts from other vibration and LED alerts the device could employ different pulsing patterns for each message type. A benefit of such notification techniques is that they have low overhead: voice commands are relatively rare and hence generating a momentary tone, vibration, or flashing light consumes little power and is arguably non-intrusive.

Unfortunately, users notoriously ignore security warning messages, as is demonstrated by numerous studies of the (in)effectiveness of warning messages in deployed systems [34, 36, 43]. There is unfortunately little reason to believe that most users would recognize and not quickly become acclimated to voice command notifications. Still, given the low cost of deploying a notification system, it may be worth considering *in combination* with some of the other defenses described below.

5.2 Defenses that challenge

There are many ways in which a device may seek confirmation from the user before executing a voice command. Devices with a screen might present a confirmation dialogue, though this limits the utility of the voice interface. We therefore consider defenses in which the user must vocally confirm interpreted voice commands. Presenting an audio challenge has the advantage of requiring the user's attention, and thus may prevent all hidden voice commands from affected the device assuming the user will not confirm an unintended command. A consistent verbal confirmation command, however, offers little protection from hidden voice commands: the attacker also provide the response in an obfuscated manner. If the attacker can monitor any random challenge provided by the device, it might also be spoofed. To be effective, the confirmation must be easily produced by the human operator and be difficult to forge by an adversary.

The Audio CAPTCHA. Such a confirmation system already exists in the form of audio CAPTCHAs [26] which is a challenge-response protocol in which the challenge consists of speech that is constructed to be difficult for computers to recognize while being easily understood by humans. The response portion of the protocol varies by the type of CAPTCHA, but commonly requires the human to transcribe the challenge.

Audio CAPTCHAs present an possible defense to hidden voice commands: before accepting a voice command, a device would require the user to correctly respond to an audio CAPTCHA, something an attacker using machine speech recognition would find difficult. While it is clear that such a defense potentially has usability issues, it may be worthwhile for commands that are damaging or difficult to undo.

Audio CAPTCHAs are useful defenses against hidden voice commands only if they are indeed secure. Previous generations of audio CAPTCHAs have been shown to be broken using automated techniques [6, 38]. As audio CAPTCHAs have improved over time [11, 27], the question arises if currently fielded audio CAPTCHAs have kept pace with improvements in speech recognition technologies. In short, they have not.

We focus our examination on two popular audio CAPTCHA systems: Google's *reCaptcha* [32] offers audio challenges initially consisting of five random digits spread over approximately ten seconds; and *NLP Captcha* [30] provides audio challenges of about three seconds each composed of four or five alphanumeric

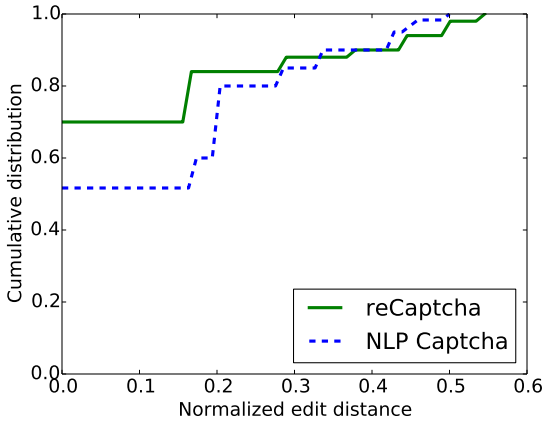

Figure 5: Accuracy of breaking audio CAPTCHA using machine based speech-to-text conversion. A normalized edit distance of zero signifies exact prediction.

characters, with the addition of the word "and" before the last character in some challenges.

We tested 50 challenges of *reCaptcha* and *NLP Captcha* each by segmenting the audio challenges before transcribing them using Google's speech recognition service. Figure 5 shows the results of transcription. Here, we show the normalized edit distance, which is the Levenshtein edit distance using characters as alphabet symbols divided by the length of the challenge. More than half and more than two-thirds of NLP Captchas and reCaptchas, respectively, are perfectly transcribed using automated techniques. Moreover, approximately 80% of CAPTCHAs produced by either system have a normalized edit distance of 0.3 or less, indicating a high frequency of at least mostly correct interpretations. This is relevant, since audio CAPTCHAs are unfortunately not easily understood by humans; to increase usability, reCaptcha provides some "leeway" and accepts almost-correct answers.

Given the ease at which they can be solved using automated techniques, the current generation of deployed audio CAPTCHA systems seems unsuitable for defending against hidden voice commands. Our results do not indicate whether or not audio CAPTCHAs are necessarily insecure. However, we remark that since computers continue to get better at speech recognition developing robust audio CAPTCHA puzzles is likely to become increasingly more difficult.

5.3 Defenses that detect and prevent

Speaker recognition. Speaker recognition (sometimes called voice authentication) has been well-explored as a biometric for authentication [7], with at least Google recently including speaker recognition as an optional feature in its Android platform [18]. Apple also introduced similar functionality in iOS [2].

However, it is unclear whether speaker verification necessarily prevents the use of hidden voice commands, especially in settings in which the adversary may be able to acquire samples of the user's voice. Existing work has demonstrated that voices may be mimicked using statistical properties[3]; for example, Aylett and Yamagishi [3] are able to mimic President George W. Bush's voice with as little of 10 minutes of his speech. Hence, it may be possible to construct an obfuscated voice command based on recordings of the user's voice that will be accepted both by the speaker recognition and the voice recognition systems. This is an interesting technical question which we defer to future work.

Importantly, speaker recognition presents three well-understood usability issues. First, a non-negligible false negative rate might limit authorized use, which provides an incentive for users to deactivate speaker recognition. Second, speaker recognition requires training, and likely necessitates the collection of a large speech corpus in order to ensure the level of accuracy necessary for reliable authentication. This need is compounded for devices such as Amazon Echo that are intended to be used by multiple users. Users may be unwilling to perform such training. Finally, speaker recognition is unsuited for applications that have no prior interactions with the user, for example, in kiosks for the visually impaired. We argue that such usability issues motivate the need for less intrusive defenses, such as the ones described next.

The "Filter". This defense decreases the fidelity of the input audio before applying speech recognition. This somewhat counterintuitive approach leverages the precision required by hidden voice commands: by slightly degrading the audio quality, normal commands are affected only slightly while obfuscated inputs, already at the cusp of being comprehensible by machine, are no longer recognized. The goal of the filter is thus to find a "sweet spot" such that a slight decrease in fidelity will not too adversely affect normal audio, but will eliminate hidden voice commands.

Our filter implementation considers audio in the time domain. For a filter rate f and an audio file consisting of s samples, we preserve $f \cdot s$ samples chosen uniformly at random and remove without replacement the remaining $(1-f)s$ samples.

We evaluate the efficacy of the filter defense using the West Point Company English speech data corpus from the University of Pennsylvania's Linguistic Data Consortium (LDC) [29]. The LDC corpus consists of American English-language speakers, 30 of them speaking the

[3] CereVoice offers an online service for "[creating] a computer version of your own voice" [8].

same 50 sentences each, and another 109 speakers each speaking a separate set of 185 sentences. For each sentence in the corpus, we construct multiple obfuscated audio files using the black-box techniques described in §3. We do not introduce any background noise to either the normal or obfuscated samples, and send all samples to Google's speech recognition system and evaluate the accuracy of its transcriptions. We tested the filter against 16,440 normal commands and 27,589 obfuscated commands, sampled uniformly at random without replacement. The audio files were in WAV format with single channel sampled at 22050 kHz with 32-bit bit depth in float data type.

We define *phoneme edit distance* δ as the Levenshtein edit distance between phoneme sequences of two transcriptions, and L as the phoneme length of unfiltered sentence. Finally we define *relative phoneme edit distance* as $\delta/(\delta+L)$, to reflect how close the two transcriptions might sound to a human listener. Figure 6 plots the cumulative distribution of the *relative phoneme edit distances* for our tested audio samples. Out of five transcripts Google returned, we pick the transcript with the smallest phoneme edit distance and calculate the *relative phoneme edit distance*.With an 80% filter (i.e., 80% of the original audio is preserved), the hidden voice command is significantly perturbed—the fraction of correctly transcribed sentences decreases from 41.74% to 7.49% when the filter is applied. However, the filter also significantly degrades normal audio, decreasing accuracy by 14.35%, which is likely unacceptable for many users who depend on voice functionality. Through some manual tuning, we find that a 92% filter offers an attractive sweet spot, causing the fraction of correctly transcribed obfuscated files to fall from 41.74% to 26.60%, while negligibly affecting normal files, whose accuracy drops only 1.06%.

The "Classifier". As a final defense we examine whether automated techniques can detect our hidden voice commands—that is, can a computer classify interpreted speech as being generated by a human or a machine? In practice, a speech recognition system could incorporate a machine learning classifier as a final processing step, dropping voice commands that are determined to be machine-generated.

We construct a hidden voice command detector using logistic regression. Our classifier uses mid-term features (mean and standard deviation) derived from short term features shown in Appendix G, extracted using the pyAudioAnalysis library [16] from the original audio signal.

We use the same LDC speech corpus as used in our filter experiments, and label each sample from the corpus as *normal*. Using the black-box procedure described in §3, we produce obfuscated hidden voice commands,

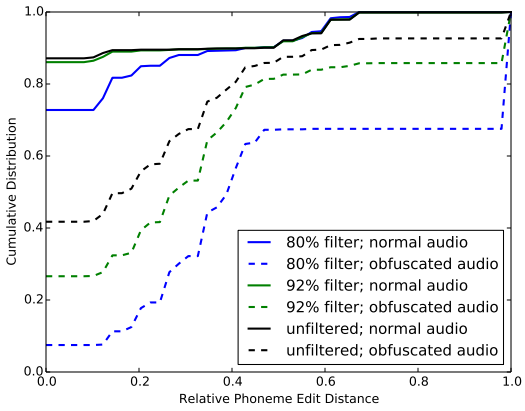

Figure 6: The cumulative distribution of relative phoneme edit distances for normal and obfuscated commands, for filtering rates of 80% and 92%.

which we label as *obfuscated*. We overlay the background noise samples described in §3.3 to mimic a noisy environment to both the *normal* and *obfuscated* samples.

We apply tenfold cross validation and report averages across the ten runs. The confusion matrix that results from classification is reported in Table 4. 99.8% of obfuscated audio were detected as such, while only incurring a 0.2% false positive rate, meaning that the classifier would incorrectly discard two out of 1000 valid commands.

To better gauge the classifier's accuracy when presented with diverse inputs, we performed additional classification tests against 51 commands generated using the white-box technique from §4 and audio from the Accent GMU dataset [42]. The Accent GMU dataset is comprised of 569 audio samples of English text spoken by different individuals with different regional accents. Neither the GMU or white-box samples were used to construct the classifier. That is, our results show the efficacy of a classifier constructed with only normal and black-box obfuscated command samples as training data. Importantly, the GMU dataset consists of all normal (non-obfuscated) samples, while the white-box dataset contains only attack commands. The confusion matrix for this classification task is presented in Table 5. For the GMU dataset, our classifier performs well, incurring less than a 1% false positive rate. The performance is worse for the white-box attack. Still, even against this strong attack which requires complete knowledge of the back-end speech recognition system, the classifier is able to flag nearly 70% of the hidden voice commands as being malicious.

Table 4: Confusion matrix of our classifier.

	Normal	Obfuscated
Normal	49.9%	0.1%
Obfuscated	0.1%	49.9%

Table 5: Confusion matrix of our classifier, when classifying audio from outside corpora.

	Normal	Attack
White-box Attack	30.7%	69.3%
Accent GMU	99.2%	0.8%

5.4 Summary of defenses

We present the first examination of defenses against hidden voice commands. Our analysis of notification defenses (§5.1) shows that security alerts are difficult to mask, but may be ignored by users. Still, given their ease of deployment and small footprint, such defenses are worth considering. Active defenses, such as audio CAPTCHAs (§5.2) have the advantage that they require users to affirm voice commands before they become effected. Unfortunately, active defenses also incur large usability costs, and the current generation of audio-based reverse Turing tests seem easily defeatable. Most promising are prevention and detection defenses (§5.3). Our findings show that filters which slightly degrade audio quality can be tuned to permit normal audio while effectively eliminating hidden voice commands. Likewise, our initial exploration of machine learning-based defenses shows that simple classification techniques yield high accuracy in distinguishing between user- and computer-generated voice commands.

6 Limitations and Discussion

While the results of our defenses are encouraging, a limitation of this paper is that the defenses do not offer proofs of security. In particular, an adversary may be able to construct hidden voice commands that are engineered to withstand filtering and defeat classifiers.

The random sampling used by our filter complicates the task of designing a "filter-resistant" hidden voice command since the adversary has no advanced knowledge of what components of his audio command will be discarded. The adversary is similarly constrained by the classifier, since the attacks we describe in §3 and §4 significantly affect the features used in classification. Of course, there might be other ways to conceal voice commands that are more resistant to information loss yet retain many characteristics of normal speech, which would likely defeat our existing detection techniques. Designing such attacks is left as a future research direction.

The attacks and accompanying evaluations in §3 and §4 demonstrate that hidden voice commands are effective against modern voice recognition systems. There is clearly room for another security arms race between more clever hidden voice commands and more robust defenses. We posit that, unfortunately, the adversary will likely always maintain an advantage so long as humans and machines process speech dissimilarly. That is, there will likely always be some room in this asymmetry for "speaking directly" to a computational speech recognition system in a manner that is not human parseable.

Future work. CMU Sphinx is a "traditional" approach to speech recognition which uses a hidden Markov model. More sophisticated techniques have recently begun to use neural networks. One natural extension of this work is to extend our white-box attack techniques to apply to RNNs.

Additional work can potentially make the audio even more difficult for an human to detect. Currently, the white-box hidden voice commands sound similar to white noise. An open question is if it might be possible to construct working attacks that sound like music or other benign noise.

7 Conclusion

While ubiquitous voice-recognition brings many benefits its security implications are not well studied. We investigate hidden voice commands which allow attackers to issue commands to devices which are otherwise unintelligible to users.

Our attacks demonstrate that these attacks are possible against currently-deployed systems, and that when knowledge of the speech recognition model is assumed more sophisticated attacks are possible which become much more difficult for humans to understand. (Audio files corresponding to our attacks are available at https://sites.google.com/site/hiddenvoicecommands16/.)

These attacks can be mitigated through a number of different defenses. Passive defenses that notify the user an action has been taken are easy to deploy and hard to stop but users may miss or ignore them. Active defenses may challenge the user to verify it is the owner who issued the command but reduce the ease of use of the system. Finally, speech recognition may be augmented to detect the differences between real human speech and synthesized obfuscated speech.

We believe this is an important new direction for future research, and hope that others will extend our analysis of potential defenses to create sound defenses which allow for devices to securely use voice-commands.

References

[1] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, and C. D. Spyropoulos. An Experimental Comparison of Naive Bayesian and Keyword-based Anti-spam Filtering with Personal e-Mail Messages. In *ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2000.

[2] Apple. Use Siri on your iPhone, iPad, or iPod touch. Support article. Available at https://support.apple.com/en-us/HT204389.

[3] M. P. Aylett and J. Yamagishi. Combining Statistical Parameteric Speech Synthesis and Unit-Selection for Automatic Voice Cloning. In *LangTech*, 2008.

[4] M. Barreno, B. Nelson, A. D. Joseph, and J. Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010.

[5] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion Attacks against Machine Learning at Test Time. In *Machine Learning and Knowledge Discovery in Databases*, 2013.

[6] E. Bursztein and S. Bethard. Decaptcha: Breaking 75% of eBay Audio CAPTCHAs. In *USENIX Workshop on Offensive Technologies (WOOT)*, 2009.

[7] J. Campbell, J.P. Speaker Recognition: A Tutorial. *Proceedings of the IEEE*, 85(9):1437–1462, 1997.

[8] CereVoice Me Voice Cloning Service. https://www.cereproc.com/en/products/cerevoiceme.

[9] Crowd Sounds — Free Sounds at SoundBible. http://soundbible.com/tags-crowd.html.

[10] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma. Adversarial Classification. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2004.

[11] M. Darnstadt, H. Meutzner, and D. Kolossa. Reducing the Cost of Breaking Audio CAPTCHAs by Active and Semi-supervised Learning. In *International Conference on Machine Learning and Applications (ICMLA)*, 2014.

[12] W. Diao, X. Liu, Z. Zhou, and K. Zhang. Your Voice Assistant is Mine: How to Abuse Speakers to Steal Information and Control Your Phone. In *ACM Workshop on Security and Privacy in Smartphones & Mobile Devices (SPSM)*, 2014.

[13] H. Drucker, S. Wu, and V. Vapnik. Support vector machines for spam categorization. *IEEE Transactions on Neural Networks*, 10(5), Sep 1999.

[14] A. Fawzi, O. Fawzi, and P. Frossard. Analysis of classifiers' robustness to adversarial perturbations. *arXiv preprint arXiv:1502.02590*, 2015.

[15] M. Fredrikson, S. Jha, and T. Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, 2015.

[16] T. Giannakopoulos. Python Audio Analysis Library: Feature Extraction, Classification, Segmentation and Applications. https://github.com/tyiannak/pyAudioAnalysis.

[17] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[18] Google. Turn on "Ok Google" on your Android. Support article. Available at https://support.google.com/websearch/answer/6031948.

[19] *Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images*, 2015. IEEE.

[20] C. Ittichaichareon, S. Suksri, and T. Yingthawornsuk. Speech recognition using MFCC. In *International Conference on Computer Graphics, Simulation and Modeling (ICGSM)*, 2012.

[21] Y. Jang, C. Song, S. P. Chung, T. Wang, and W. Lee. A11y Attacks: Exploiting Accessibility in Operating Systems. In *ACM Conference on Computer and Communications Security (CCS)*, November 2014.

[22] A. Kantchelian, S. Afroz, L. Huang, A. C. Islam, B. Miller, M. C. Tschantz, R. Greenstadt, A. D. Joseph, and J. D. Tygar. Approaches to Adversarial Drift. In *ACM Workshop on Artificial Intelligence and Security*, 2013.

[23] C. Kasmi and J. Lopes Esteves. Iemi threats for information security: Remote command injection on modern smartphones. *IEEE Transactions on Electromagnetic Compatibility*, PP(99):1–4, 2015.

[24] P. Lamere, P. Kwok, W. Walker, E. Gouvea, R. Singh, B. Raj, and P. Wolf. Design of the CMU Sphinx-4 Decoder. In *Eighth European Conference on Speech Communication and Technology*, 2003.

[25] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *Conference on Computer Vision and Pattern Recognition (CVPR) 2015*, 2015.

[26] M. May. Inaccessibility of CAPTCHA: Alternatives to Visual Turing Tests on the Web. Technical report, W3C Working Group Note, 2005. Available at http://www.w3.org/TR/turingtest/.

[27] H. Meutzner, S. Gupta, and D. Kolossa. Constructing Secure Audio CAPTCHAs by Exploiting Differences Between Humans and Machines. In *Annual ACM Conference on Human Factors in Computing Systems (CHI)*, 2015.

[28] D. E. Meyer and R. W. Schvaneveldt. Facilitation in Recognizing Pairs of Words: Evidence of a Dependence between Retrieval Operations. *Journal of Experimental Psychology*, 90(2):227, 1971.

[29] J. Morgan, S. LaRocca, S. Bellinger, and C. C. Ruscelli. West Point Company G3 American English Speech. Linguistic Data Consortium, item LDC2005S30. University of Pennsylvania. Available at https://catalog.ldc.upenn.edu/LDC2005S30, 2005.

[30] NLP Captcha. http://nlpcaptcha.in/.

[31] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. *arXiv preprint arXiv:1511.07528*, 2015.

[32] reCAPTCHA. http://google.com/recaptcha.

[33] H. Sak, A. Senior, K. Rao, F. Beaufays, and J. Schalkwyk. Google Voice Search: Faster and More Accurate, 2015. Google Research Blog post. Available at http://googleresearch.blogspot.com/2015/09/google-voice-search-faster-and-more.html.

[34] S. Schechter, R. Dhamija, A. Ozment, and I. Fischer. The Emperor's New Security Indicators: An Evaluation of Website Authentication and the Effect of Role Playing on Usability Studies. In *IEEE Symposium on Security and Privacy (Oakland)*, 2007.

[35] R. Schlegel, K. Zhang, X.-y. Zhou, M. Intwala, A. Kapadia, and X. Wang. Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones. In *Network and Distributed System Security Symposium (NDSS)*, 2011.

[36] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor. Crying Wolf: An Empirical Study of SSL Warning Effectiveness. In *USENIX Security Symposium (USENIX)*, 2009.

[37] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing Properties of Neural Networks. *arXiv preprint arXiv:1312.6199*, 2013.

[38] J. Tam, J. Simsa, S. Hyde, and L. V. Ahn. Breaking Audio CAPTCHAs. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

[39] J. Tygar. Adversarial Machine Learning. *IEEE Internet Computing*, 15(5):4–6, 2011.

[40] T. Vaidya, Y. Zhang, M. Sherr, and C. Shields. Cocaine Noodles: Exploiting the Gap between Human and Machine Speech Recognition. In *USENIX Workshop on Offensive Technologies (WOOT)*, August 2015.

[41] O. Viikki and K. Laurila. Cepstral Domain Segmental Feature Vector Normalization for Noise Robust Speech Recognition. *Speech Communication*, 25(13):133–147, 1998.

[42] S. H. Weinberger. Speech Accent Archive. George Mason University, 2015. Available at http://accent.gmu.edu.

[43] M. Wu, R. C. Miller, and S. L. Garfinkel. Do Security Toolbars Actually Prevent Phishing Attacks? In *SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2006.

A Additional Background on Sphinx

As mentioned in §4, the first transform taken by Sphinx is to split the audio in to overlapping frames, as shown in Figure 7. In Sphinx, frames are 26ms (410 samples) long, and a new frame begins every 10ms (160 samples).

MFC transform. Once Sphinx creates frames, it runs the MFC algorithm. Sphinx's MFC implementation involves five steps:

1. **Pre-emphasizer:** Applies a high-pass filter that reduces the amplitude of low-frequencies.
2. **Cosine windower:** Weights the samples of the frame so the earlier and later samples have lower amplitude.
3. **FFT:** Computes the first 257 terms of the (complex-valued) Fast Fourier Transform of the signal and returns the squared norm of each.

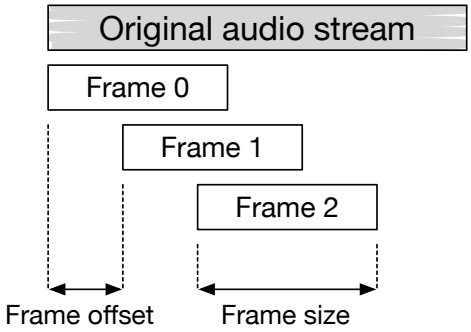

Figure 7: The audio file is split into overlapping frames.

4. **Mel filter:** Reduces the dimensionality further by splitting the 257 FFT terms into 40 buckets, summing the values in each bucket, then returning the log of each sum.
5. **DCT:** Computes the first 13 terms of the Discrete Cosine Transform (DCT) of the 40 bucketed values.[4]

Despite the many steps involved in the MFC pipeline, the entire process (except the running average and derivatives steps) can be simplified into a single equation:

$$\mathrm{MFCC}(x) = C\log(B\,\|Ax\|^2)$$

where the norm, squaring and log are done componentwise to each element of the vector. A is a 410×257 matrix which contains the computation performed by the pre-emphasizer, cosine windower, and FFT. B is a 257×40 matrix which computes the Mel filter, and C is a 40×13 matrix which computes the DCT.

Sphinx is configured with a dictionary file, which lists all valid words and maps each word to its phonemes, and a grammar file, which specifies a BNF-style formal grammar of what constitutes a valid sequence of words. In our experiments we omit the grammar file and assume any word can follow any other with equal probability. (This makes our job as an attacker more difficult.)

The HMM states can be thought of as phonemes, with an edge between two phonemes that can occur consecutively in some word. Sphinx's model imposes additional restrictions: its HMM is constructed so that all paths in the HMM correspond to a valid sequence of words in the dictionary. Because of this, any valid path through the HMM corresponds to a valid sequence of words. For example, since the phoneme "g" never follows itself, the HMM only allows one "g" to follow another if they are the start and end of words, respectively.

[4]While it may seem strange to take the DCT of the frequency-domain data, this second FFT is able to extract higher-level features about which frequencies are common, and is more tolerant to a change in pitch.

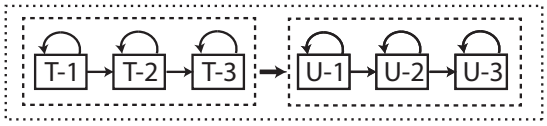

Figure 8: The HMM used by Sphinx encoding the word "two". Each phoneme is split into three HMM states (which may repeat). These HMM states must occur in sequence to complete a phoneme. The innermost boxes are the phoneme HMM states; the two dashed boxes represent the phoneme, and the outer dashed box the word "two".

The above description is slightly incomplete. In reality, each phoneme is split into three HMM states, which must occur in a specific order, as shown in Figure 8. Each state corresponds to the beginning, middle, or end of a phoneme. A beginning-state has an edge to the middle-state, and the middle-state has an edge to the end-state. The end-phoneme HMM state connects to beginning-phoneme HMM states of other phonemes. Each state also has a self-loop that allows the state to be repeated.

Given a sequence of 39-vectors, Sphinx uses the Viterbi algorithm to try to find the 100 most likely paths through the HMM model (or an approximation thereto).

B Detailed Machine Comprehension of Black-box Attack

The detailed results of machine comprehension of black-box attacks are presented in Figure 9.

We note that Figure 9 contains an oddity: in a few instances, the transcription success rate decreases as the SNR increases. We suspect that this is due to our use of median SNR, since the background samples contain non-uniform noise and transient spikes in ambient noise levels may adversely affect recognition. Overall, however, we observe a clear (and expected) trend in which transcription accuracy improves as SNR increases.

C Analysis of Transcriptions using Phoneme-Based Edit Distance Metrics

C.1 Black-box attack

To verify the results of the white-box survey and to better understand the results of Amazon Mechanical Turk Study, we first performed a simple binary classification of transcription responses provided by Turk workers.

We define *phoneme edit distance* δ as the Levenshtein edit distance between phonemes of two transcriptions. We define ϕ as δ/L, where L is the phoneme length of normal command sentence. The use of ϕ reflects how close the transcriptions might sound to a human listener. $\phi < 0.5$ indicates that the human listener successfully

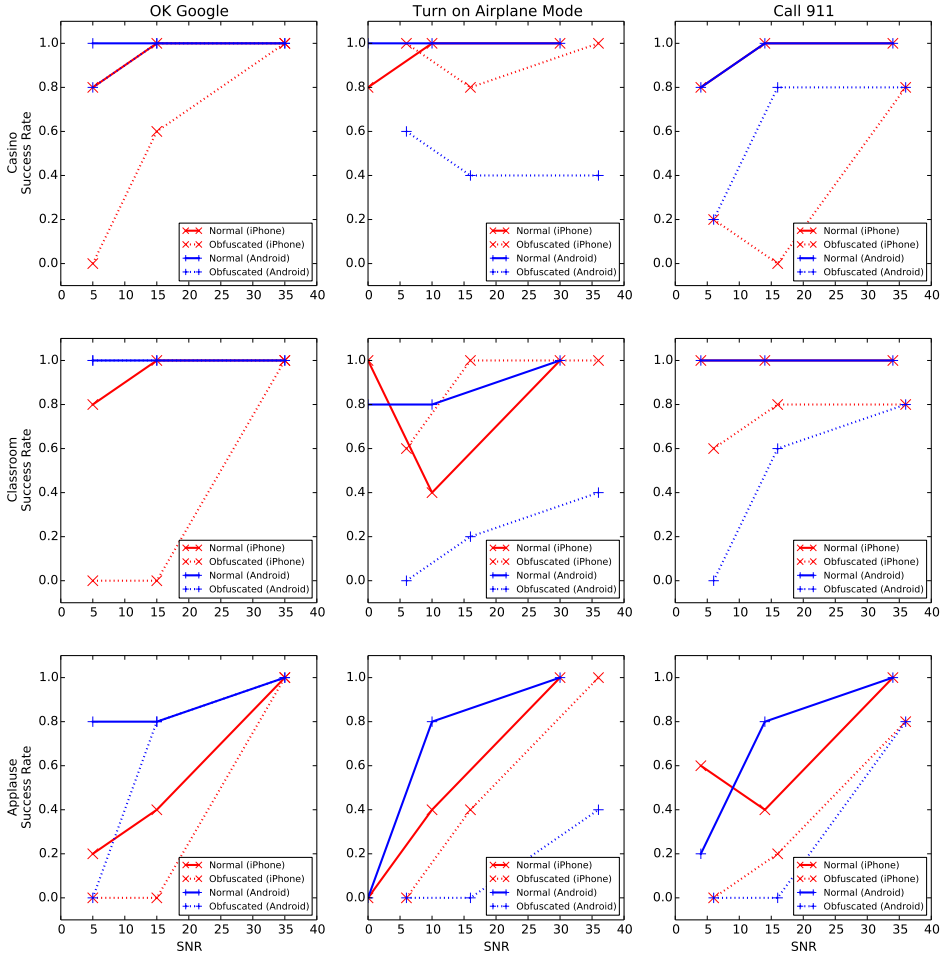

Figure 9: Machine understanding of normal and obfuscated variants of "OK Google", "Turn on Airplane Mode", and "Call 911" voice commands *(column-wise)* under different background noises *(row-wise)*. Each graph shows the measured average success rate (the fraction of correct transcripts) on the y-axis as a function of the signal-to-noise ratio.

comprehended at least 50% of the underlying voice command. We consider this as successful comprehension by human, implying attack failure; otherwise, we consider it a success for the attacker. Table 6 shows the results of our binary classification. The difference between the success rates of normal and obfuscated commands is similar to that of the human listeners in Table 2, validating the survey results.

We used *relative phoneme edit distance* to show the gap between transcriptions of normal and obfuscated commands submitted by turk workers. The *relative phoneme edit distance* is calculated as $\delta/(\delta + L)$, L is again the phoneme length of normal command sentence. The relative phoneme edit distance has a range of $[0, 1)$, where 0 indicates exact match and larger relative phoneme edit distances mean the evaluator's transcription further deviates from the ground truth. By this definition, a value of 0.5 is achievable by transcribing silence. Values above 0.5 indicate no relationship between the transcription and correct audio.

Figure 10 shows the cumulative distribution of the relative phoneme edit distance for the *(left)* "OK Google", *(center)* "Turn on Airplane Mode" and *(right)* "Call 911" voice commands. These graphs show similar results as reported in Table 2: Turk workers were adept at correctly transcribing the normal commands even in presence of

Table 6: Black-box attack. Percentages show the fraction of human listeners who were able to comprehend at least 50% of voice commands.

	OK Google	**Turn On Airplane Mode**	**Call 911**
Normal	97% (97/100)	89% (102/114)	92% (75/81)
Obfuscated	24% (23/94)	47% (52/111)	95% (62/65)

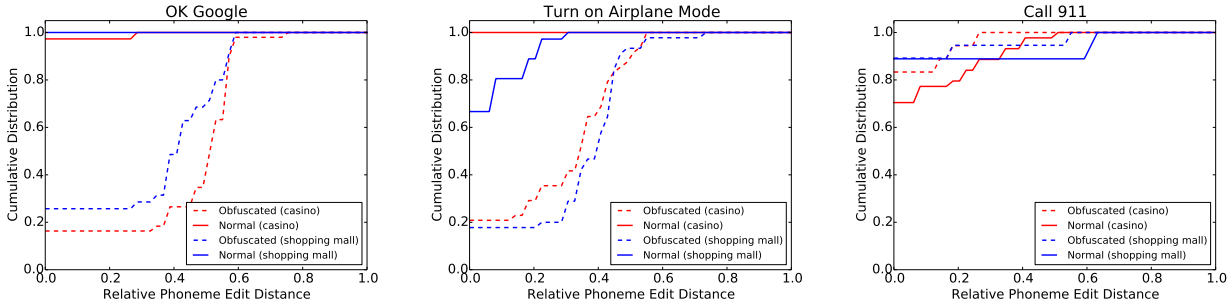

Figure 10: Cumulative distribution of relative phoneme edit distances of Amazon Mechanical Turk workers' transcriptions for *(left)* "OK Google", *(center)* "Turn on Airplane Mode" and *(right)* "Call 911" voice commands, with casino and shopping mall background noises. The attack is successful for the first two commands, but fails for the third.

Table 7: White-box attack. Percentages show the fraction of human listeners who were able to comprehend at least 50% of phonemes in a command.

	Command
Normal	97% (297/310)
Obfuscated	10% (37/377)

background noise; over 90% of workers made perfect transcriptions with an edit distance of 0. However, the workers were far less able to correctly comprehend the obfuscated commands: less than 30% of workers were able to achieve a relative edit distance less than 0.2 for the "OK Google" and "Turn on Airplane Mode" commands.

C.2 White-box attack

To again verify the results of our authors review of the turk study, we computed the edit distance of transcribed commands with actual commands. Table 7 again says a command is a match if at least 50% of phonemes were transcribed correctly, to eliminate potential author bias. As can be seen, this metric is less strict both for normal commands and obfuscated commands, but the drop in quality is nearly as strong.

D Canceling out the Beep

Even when constrained to simplistic and conservative mathematical models, it is difficult to cancel out a beep played by a mobile device.

D.1 Two ears difficulties

Setup: The victim has two ears located at points E and F, and a device at point P. The attacker has complete control over a speaker at point A.

Threat model: The attacker has complete knowledge of the setup, including what the beep sounds like, when the beep will begin playing, and the location of all four points E, F, P and A. We assume for simplicity that sound amplitude does not decrease with distance.

The attacker loses the game if the victim hears a sound in either ear. Our question, then, is: *can the attacker cancel out the sound of the beep in both ears simultaneously*? Since sound amplitude does not attenuate with distance, the attacker can focus solely on phase matching: to cancel out a sound, the attacker has to play a signal that is exactly π radians out of phase with the beep. This means the attacker has to know the phase of the signal to a good degree of accuracy.

In our model, canceling out sound at one ear (say E) is easy for the attacker. The attacker knows the distance d_{PE}, and so knows t_{PE}, the time it will take for the sound to propagate from P to E. Similarly, the attacker knows t_{AE}. This is enough to determine the delay that he needs to introduce: he should start playing his signal $\frac{(d_{PE}-d_{AE}) \pmod{\lambda}}{c}$ (where λ is the wavelength) seconds after the start of the beep (where c is the speed of sound), and the signal he should play from his speaker is the inverse of the beep (an "anti-beep").

However, people usually have two ears, and so there will still be some remnant of the beep at the other ear F.

In particular, the beep will arrive at that ear $\frac{d_{PF}}{c}$ seconds after being played, while the anti-beep will arrive $\frac{d_{AF}}{c}$ seconds after the anti-beep starts, i.e., $\frac{d_{PE}-d_{AE}+d_{AF}}{c}$ seconds after the beep starts. This means that the anti-beep will be delayed by $\frac{d_{PE}-d_{AE}+d_{AF}-d_{PF}}{c}$ seconds compared to the beep.

Therefore, the attacker must be sure that they are placed exactly correctly so that the cancellation occurs at just the right time for both ears. This is the set of points where $(d_{PE}-d_{AE}+d_{AF}-d_{PF})=0$. That is, the attacker can be standing anywhere along half of a hyperbola around the user.

Finally, there is one more issue: any device which can perform voice recognition must have a microphone, and so can therefore listen actively for an attack. This then requires not only that the attacker be able to produce exactly the inverse signal at both ears, but also zero total volume at the device's location. This then fixes the attacker's location to only one potential point in space.

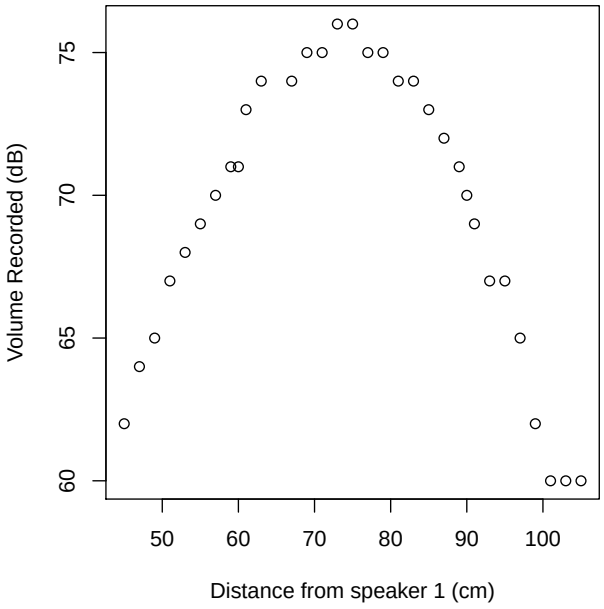

Figure 11: Plot of the amplitude of attempted noise cancellation of a tone at 440Hz

D.2 Real-world difficulties

In the above setup we assumed a highly idealized model of the real world. For instance, we assumed that the attacker knows all distances involved very precisely. This is of course difficult to achieve in practice (especially if the victim moves his head). Our calculations show that canceling over 90% of the beep requires an error of at most 3% in the phase. Putting this into perspective, for a 1Khz beep, to eliminate 90% of the noise, the adversary needs to be accurate to within 3 inches.

In practice, the attack is even more difficult than described above. The adversary may have to contend with multiple observers, and has to consider background noise, amplitude attenuation with distance, and so on.

Even so, to investigate the ability of an attacker to cancel sound in near-ideal conditions, we conducted an experiment to show how sound amplitude varies as a function of the phase difference in ideal conditions. The setup is as follows: two speakers are placed facing each other, separated by a distance d. Both speakers play the same pure tone at the same amplitude. We placed a microphone in between, and measured the sound amplitude at various points on the line segment joining the two. For our experiment, $d = 1.5$m and the frequency of the tone is $f = 440$Hz. The results are plotted in Figure 11.

As can be seen, the total cancellation does follow a sine wave as would be expected, however there is noise due to real-world difficulties. This only makes the attacker's job more difficult.

E Least Squares Algorithm

Below, we present the least squares algorithm that is used in our improved white box attack presented in §4.4.

1. Define A as the $39k \times 13(6+k)$ dimensional matrix which computes the derivative of a sequence of $6+k$ 13-vectors and returns the k resulting 39-vectors.
2. Define b as the $39k$ dimensional vector corresponding to the concatenation of the $k-1$ 39-vectors in $\bar{g}$ and the single 39-vector g.
3. Split A in two pieces, with A_L being the left $13k$ columns, and A_R being the right 6×13 columns.
4. Define $\bar{f}$ as the concatenation of the 13-vectors in f.
5. Define $\bar{b} = b - A_L \cdot \bar{f}$.
6. Using least squares, find the best approximate solution $\hat{x}$ to the system of equations $A_r \cdot \hat{x} = \bar{b}$.
7. Return $(|(A_r \cdot \hat{x}) - \bar{b}|, \hat{x})$

F Machine Interpretation of Obfuscated Command

Table 8: For each of the three phrases generated in our white-box attack, the phrase that Sphinx recognized. This data is used to alter the lengths of each phoneme to reach words more accurately. Some words such as "for" and "four" are pronounced exactly the same: Sphinx has no language model and so makes errors here.

Phrases as recognized by CMU Sphinx	
Count	Phrase
3	okay google browse evil dot com
1	okay google browse evil that come
1	okay google browse evil them com
1	okay google browse for evil dot com
6	okay google browse two evil dot com
2	okay google browse two evil that com
1	okay google browse who evil not com
1	okay google browse who evil that com
1	okay up browse evil dot com
5	okay google picture
2	okay google take a picture
1	okay google take of
1	okay google take of picture
6	okay google take picture
10	okay google text one three for five
1	okay google text one two three for five
2	okay google text one who three for five
3	okay google text want three for five

G Short-Term Features used by Classifier Defense

Table 9: Short term features used for extracting mid-term features.

Feature	Description
Zero Crossing Rate	The rate of sign-changes of the signal during the duration of a particular frame.
Energy	The sum of squares of the signal values, normalized by the respective frame length.
Entropy of Energy	The entropy of sub-frames' normalized energies.
Spectral Centroid	The center of gravity of the spectrum.
Spectral Spread	The second central moment of the spectrum.
Spectral Entropy	Entropy of the normalized spectral energies for a set of sub-frames.
Spectral Flux	The squared difference between the normalized magnitudes of the spectra of the two successive frames.
Spectral Rolloff	The frequency below which 90% of the magnitude distribution of the spectrum is concentrated.
MFCCs	Mel Frequency Cepstral Coefficients
Chroma Vector	A 12-element representation of the spectral energy
Chroma Deviation	The standard deviation of the 12 chroma coefficients.