EXPLOITING AND HARNESSING THE PROCESSES AND DIFFERENCES OF SPEECH UNDERSTANDING IN HUMANS AND MACHINES

A Dissertation
submitted to the Faculty of the Graduate School of Arts and Sciences of Georgetown University in partial fulfillment of the requirements of the degree of Doctor of Philosophy in Computer Science

By

Tavish Vaidya, M.S.

Washington, DC
July 19, 2019
Humans have long communicated verbally with one another in various languages. Voice can be considered the most convenient way of communicating face-to-face, either in-person or virtually, with technology’s assistance. The ease of use and familiarity of verbal interactions makes voice the natural choice for providing input to machines for certain types of interactions that do not require long inputs from humans, e.g., providing instructions for actions or asking questions, etc. Such interactions have become popular, given the widespread adoption of personal voice assistant technologies in handheld devices, wearables and in-home assistants. Be it asking for today’s weather or making a phone call, setting a reminder or buying a product from Amazon, voice assistants have made interactions with machines seamless.

Successful voice communication, either between humans or humans and machines, requires two processes to succeed: 1) the speaker must speak a phrase in some language, and 2) the listener(s) must be able to hear the spoken phrase and be able to recognize and understand that language. Crucially, the processes by which humans understand speech and by which machines understand speech are different. This thesis exploits this gap in the mechanisms of understanding of audio between humans and machines to demonstrate novel attacks against voice command interfaces and to propose defenses against them.
Since humans and machines understand speech differently, this thesis harnesses this difference to improve defenses against attacks on voice traffic between machines. An important aspect of voice interactions is privacy. In a face-to-face setting, there is a reasonable expectation that only the listeners of the conversation are the communicators themselves. The same privacy guarantees are expected in a virtual setting, achievable through the use of encryption to provide confidentiality guarantees for voice data in transit. However, previous works have shown attacks on encrypted voice communications that allow an attacker to infer attributes (e.g., the language being spoken, the identities of the speakers, etc.) about the underlying audio without needing to decrypt the communications. This thesis proposes a unilateral defense that leverages the process of human understanding of audio to improve existing defenses against these known attacks on encrypted VoIP streams and does not require any participation from the receiver.

This thesis also investigates the emerging privacy challenges associated with the collection of users’ voice data. The popularity of personal voice assistants can also be attributed to their high accuracy in correctly understanding users’ voice inputs. Such assistants record the voice inputs and send them to the cloud for processing the requests. Often, these requests are stored by the service providers to be later used for improving their systems and services. Also, the raw audio from these requests may also be shared with third parties. This has important consequences for privacy. A person’s voice is considered his fingerprint and protecting this information is vital.

The final component of this thesis discusses various developments in speech synthesis that threaten user privacy. We propose privacy-preserving defenses that protect users’ voice profiles, enabling them to use voice interfaces while limiting the exposure of their voice characteristics and fingerprints.
ACKNOWLEDGMENTS

This thesis could not have been completed without the support of many people throughout this journey. I will try to mention a few and apologize in advance for anyone I might have inadvertently missed.

My journey at Georgetown would not have started if it wasn’t for my father, Krishan, who suggested I consider applying to Georgetown for Ph.D. His guidance and wisdom have immensely helped me in every aspect of my life. Thank you to my mother, Mukta, for all her love, care and affection. It would have undoubtedly been an impossible journey without her blessings.

A special thanks to Micah for being such an awesome advisor and for taking me as his third student advisee. Without his guidance, I most likely would have never got to the point of finishing this thesis. In addition to the invaluable research input and feedback, Micah has always motivated and encouraged me to come up with new research ideas and pursue my research interests. I have learned a lot from Micah over the past years and I am deeply indebted to him.

I would also like to thank Professors Boon Thau Loo, Eric Burger, Clay Shields and Wenchao Zhou for being on my committee and for their time and support. Their suggestions and comments have greatly helped in the creation and completion of this thesis.

Thank you to the many members of the CS department at Georgetown for being wonderful friends. The journey wouldn’t have been the same without the stress-busting conversations with Yuankai, Luca, Arman, Mohammad, Katina and Sean.

Thank you to my relatives in the United States for being my family away from home. My aunt Manju and uncle Parmendar have been a constant source of encour-
agement, motivation and affection. I will forever miss the positivity, calmness and peace that flowed from my uncle, who suddenly passed away a few months ago.

Finally, thank you to my wonderful partner Iswerya for always being there. It is hard to imagine getting here without her love, patience and support. She has been an unwavering source of confidence and strength and continues to inspire me everyday.

The research conducted in the process of developing this thesis was partially supported by NSF award CNS-1718498 and DARPA under Contract No. HR0011-16-C-0056. The opinions, findings, and conclusions or recommendations expressed in this work are strictly those of the author and do not necessarily reflect the official policy or position of any employer or funding agency.
# Table of Contents

1 Introduction ................................................................. 1
   1.1 Human Understanding of Speech .................................. 3
   1.2 Machine Understanding of Speech ................................. 5
   1.3 Exploiting Differences in Audio Understanding ............... 7
   1.4 Harnessing Differences in Audio Understanding ............... 8
   1.5 Harnessing the Process of Audio Understanding in Humans ... 8
   1.6 Addressing Privacy Challenges ..................................... 9
   1.7 Research Questions ................................................... 11
   1.8 Contributions .......................................................... 12
   1.9 Organization ............................................................ 13

2 Background and Related Work ........................................... 15
   2.1 Speech Recognition Overview ....................................... 15
   2.2 Mel-frequency Cepstrum Coefficients ............................. 17
   2.3 Machine Learning Models for Speech Recognition ............. 18
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4</td>
<td>Attacks Against Machine Learning Systems</td>
<td>19</td>
</tr>
<tr>
<td>2.5</td>
<td>Attacks on Voice Interfaces</td>
<td>24</td>
</tr>
<tr>
<td>2.6</td>
<td>Signal Pattern Matching</td>
<td>27</td>
</tr>
<tr>
<td>2.7</td>
<td>Defenses for Voice Interfaces</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>Attacks Against Voice Interfaces</td>
<td>29</td>
</tr>
<tr>
<td>3.1</td>
<td>Attack Utility</td>
<td>30</td>
</tr>
<tr>
<td>3.2</td>
<td>Black-box Attack</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>White-Box Attack</td>
<td>46</td>
</tr>
<tr>
<td>3.4</td>
<td>Summary</td>
<td>53</td>
</tr>
<tr>
<td>4</td>
<td>Defenses Against Hidden Voice Commands</td>
<td>54</td>
</tr>
<tr>
<td>4.1</td>
<td>Defenses that Notify</td>
<td>55</td>
</tr>
<tr>
<td>4.2</td>
<td>Defenses that Challenge</td>
<td>56</td>
</tr>
<tr>
<td>4.3</td>
<td>Defenses that Detect and Prevent</td>
<td>58</td>
</tr>
<tr>
<td>4.4</td>
<td>Summary of Defenses</td>
<td>64</td>
</tr>
<tr>
<td>5</td>
<td>Whisper: A Novel Defense Against VoIP Traffic Re-Identification Attacks.</td>
<td>66</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>67</td>
</tr>
<tr>
<td>5.2</td>
<td>Background on VoIP Attacks and Defenses</td>
<td>71</td>
</tr>
<tr>
<td>5.3</td>
<td>User and Attack Models</td>
<td>73</td>
</tr>
<tr>
<td>5.4</td>
<td>Methodology</td>
<td>75</td>
</tr>
<tr>
<td>5.5</td>
<td>Evaluation</td>
<td>86</td>
</tr>
<tr>
<td>5.6</td>
<td>Tuning Whisper</td>
<td>96</td>
</tr>
<tr>
<td>5.7</td>
<td>Summary</td>
<td>100</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Central auditory pathway and auditory cortex in human brain. 4
1.2 Overview of a typical speech recognition system. 5
3.1 Adversary’s workflow for producing an obfuscated audio command from a normal command for black-box attack. 33
3.2 Machine understanding of normal and obfuscated variants of “OK Google”, “Turn on Airplane Mode”, and “Call 911” voice commands under Mall background noise. 40
3.3 Cumulative distribution of relative phoneme edit distances of Amazon Mechanical Turk workers’ transcriptions for “OK Google”, “Turn on Airplane Mode” and “Call 911” voice commands, with casino and shopping mall background noises. 45
3.4 Adversary’s workflow for producing an obfuscated audio command for a target text for the white-box attack. 48
4.1 Accuracy of breaking audio CAPTCHA using machine based speech-to-text conversion. 59
4.2 The cumulative distribution of relative phoneme edit distances for normal and obfuscated commands, for filtering rates of 80% and 92%. 62
5.1 Overview of a traffic re-identification attack on an encrypted VoIP stream. 68
5.2 Overview of Whisper. 69
5.3 Whisper’s workflow of modifying an input audio frame using tuning audio. 83
5.4 CDF of ViSQOL quality scores for baseline audio with no encoding (“No Codec”) and audio encoded with the Opus codec without (“Opus”) and with (“Whisper”) the Whisper defense. 91
5.5 Sections and flow of the user study. 93
5.6 Audio quality as reported by human evaluators for audio with no encoding and audio encoded with the Opus codec with and without Whisper defense. 95
5.7 Effect of fallback options on various performance metrics. 97
5.8 Effect of different pktSizeWeights values on various performance metrics. 98
5.9 Effect of strictness parameter on performance metrics. 99
6.1 Impact of audio sanitizer on transcription accuracy. 114
6.2 Sections and flow of the user study. 116
List of Tables

3.1 Mel-frequency cepstral coefficients parameters tuned to produce obfuscated audio. ........................................ 35
3.2 Black-box attack results. .................................................. 39
3.3 Percentages of human listeners who were able to correctly comprehend at least 50% of the black-box attack voice commands. .................................................. 44
3.4 White-box attack results. .......................................................... 52
4.1 Confusion matrix of our classifier. .............................................. 63
4.2 Confusion matrix of our classifier, when classifying audio from outside corpora. .................................................. 63
5.1 Participant demographics for the user study to determine audio quality.. .................................................. 94
5.2 Number of responses for each type of audio depending on how the audio was encoded. .................................................. 94
6.1 Modifications performed to various features by the audio sanitizer. . . 113
6.2 Number of participants assigned to each baseline condition and each of the four test conditions. .................................................. 118
6.3 Participant demographics for the user study to determine the efficacy of audio sanitizer. .................................................. 119
6.4 Summary of responses from the user study for various conditions. . . 120
Chapter 1

Introduction

Over the past few years, voice input has gained immense popularity as a way of interacting with machines. This is evident from the ubiquitous presence of voice assistants on smart devices: Siri on iOS, Google Assistant on Android, Alexa on Amazon devices, Cortana on Microsoft platforms, etc. Voice assistants are becoming increasingly popular. Wearables, smartphones, in-home assistants, IoT devices, tablets and laptops are some of many examples of types of devices that support voice input, and this list is very likely going to increase in the near future. Just as we are used to talking to one another, humans have natural familiarity with using voice interactions for communication, which undoubtedly makes voice the top choice to allow humans to seamlessly talk to machines. Unlike traditional touch and type inputs, interaction over a voice channel eliminates the need of physical interaction with devices for providing input. Consequently, devices that are either limited by their surface area for physical touch input (e.g., wearables) or feature hands-free user interactions
(e.g., in-home assistants such as Google Home and Amazon Echo) can extensively leverage voice input. The broadcast nature of voice also facilitates user interactions with voice-enabled devices from a distance, further enriching user experience.

Today, humans interact with devices over voice channels in their day-to-day activities ranging from talking to their smartphones, tablets running Siri, Google Assistant, etc., to interacting with in-home voice assistants such as Amazon Alexa, Google Home, etc. to play their favorite artist or to put an item in their shopping list. Apart from being capable of answering questions, setting reminders, playing music, etc., in-home voice assistants also enable users to control various IoT devices with voice commands. Amazon Alexa, Google Home and Apple Home Kit all support integration with IoT devices such as smart lights, smart switches, thermostats, home security systems, etc., all of which can then be controlled by a simple utterance of a voice command. Other voice enabled devices finding ways into human lives include voice enabled alarm clocks, smartwatches, smart TVs, vacuum cleaners, etc. Devices accepting voice inputs are already ubiquitous, and the number and types of devices accepting voice as input is only going to increase further with improvements in voice understanding by machines.

Successful voice communication, either between humans or humans and machines, requires two processes to succeed: 1) the speaker must speak a phrase in some language, and 2) the listener(s) must be able to hear the spoken phrase and be able to recognize and understand that language. In recent years, the ability of machines to understand audio has significantly improved. The rising popularity of personal voice assistants and the adoption of voice as an input to machines can be directly attributed to the high accuracy of machines being able to understand users’ voice inputs. Improvements in speech recognition algorithms coupled with advances in hardware for faster processing of machine learning algorithms have made it possible
for machines to transcribe audio in realtime with minimal errors. The result of these advances have made it possible for humans to (mostly) seamlessly communicate with machines. A user is able to ask questions to a machine verbally and the machine is able to provide a response over the voice channel. A successful dialogue requires both the user and machine to first understand the spoken audio. The mechanisms of processing and understanding of audio, however, differs in humans and machines.

This thesis leverages these differences in processes of audio understanding between humans and machines to develop a new science of hidden voice commands. We consider the effects of hidden voice commands from multiple perspectives: as an attacker who wishes to perform unauthorized actions and as a defender who wants to thwart such attacks. We also investigate how these differences in understanding can be used to strengthen the security of existing systems. Finally, we describe and address the privacy challenges associated with collection and storage of vast amounts of voice data.

1.1 Human Understanding of Speech

The process of understanding audio in humans is complex and has been studied as a two-step process: speech perception and language processing. Speech perception is the ability of humans to hear and understand speech while language processing deals with how humans communicate their ideas and how the brain processes and makes sense of these communications. We briefly describe the process of speech understanding in humans at a high level.

The first step in speech understanding in humans is the conversion of audio signals into electrical impulses. Sound waves enter the ear canal and cause vibrations in the eardrum which are amplified by three tiny bones in the middle ear. The
amplified vibrations are then sent to cochlea, where chemical reactions triggered by the vibrations create electrical signals that are forwarded to the brain via the auditory nerve for understanding [26].

The electrical signals end up in the primary auditory cortex located in the temporal area of the human brain after traveling through the auditory pathway, as shown in Figure 1.1. Different features of the audio such as intensity, frequency, etc. are analyzed by different parts of the auditory pathway. Sounds that contain speech follow a different pathway which is unique to humans. Various regions of the brain in this pathway perform multiple functions ranging from understanding sounds using acoustic-phonetic features of the signal to final comprehension of speech by decoding the meaning of individual words and understanding what the words mean together in a sentence [11, 74, 79, 105, 112, 114].
1.2 Machine Understanding of Speech

Understanding of speech by machines can be neatly broken into two separate processes: a) Speech Recognition, the process of converting spoken audio into corresponding text representation and b) Natural Language Processing, the process of understanding the meaning of a language usually represented as text. This thesis focuses on the speech recognition part of audio understanding by machines while the natural language processing part falls outside its scope. Note that we will use the terms *speech recognition* and *machine understanding* interchangeably to refer to the process of converting audio to text by machines throughout this thesis.

Speech recognition in machines tries to approximate the process of audio understanding in the human brain, but remains quite different from the natural process. Figure 1.2 shows a high-level overview of a typical speech recognition system, which consists of the following four steps: pre-processing, feature extraction, model-based prediction, and post-processing.

Pre-processing 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 \[101, 153\]. MFC transform 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 more recent systems use recurrent neural networks (RNNs) for predicting the spoken audio. Finally, a post-processing step ranks the text predictions by employing additional sources of information, such as grammar rules or locality of words.

1.2.1 Differences in Audio Understanding

Humans and machines understand audio in very different ways. While the accuracy of understanding audio by machines is reaching parity with humans \[18, 165\], there exists a gap with respect to what audio machines can understand and what humans can understand because of the differences in their respective processes of audio understanding. This thesis exploits this gap in the mechanisms of understanding of audio between humans and machines to demonstrate novel attacks against voice command interfaces and proposes defenses against them.
1.3 Exploiting Differences in Audio Understanding

Voice interfaces are just another input medium to provide commands to a machine. Many devices function in always-on mode, in which they continuously listen for possible voice input commands. These commands can be any command that the device understands. They could be innocuous commands such as setting an alarm, reducing the output volume, etc., or commands that could potentially be misused, e.g., calling a phone number, unlocking the front door, transferring money, etc. With the absence of any security mechanisms to identify the speaker of the audio input, devices accepting voice input cannot discriminate whether the source of audio command is authorized to provide input. Therefore, anyone within the proximity of the device can issue a voice command and trigger a functionality. The severity of this depends upon what commands are supported and accepted by the targeted device.

In many scenarios, this is not very interesting from an attacker’s perspective since the owner of the device could recognize the unauthorized input and take appropriate actions to prevent or minimize the desired effect of the unauthorized input command. A more clever way of carrying out such an attack would be to produce audio that is only recognized by the target device but not by human listeners.

A key contribution of this thesis is exploiting the gap in machine and human understanding of audio to generate such audio and demonstrate novel attacks on voice input interfaces. By varying the amount of acoustic information in the audio, we can selectively choose to make audio harder to comprehend by humans without affecting the ability of machines to understand the same audio. As we show later in Chapter 3, such selectively generated audio can be used to command voice assistants
and voice-enabled smart devices to carry out actions while being incomprehensible to the human listener.

1.4 Harnessing Differences in Audio Understanding

This thesis additionally discusses and proposes new defenses that harness the gap in audio understanding between humans and machines, to protect voice interfaces against the proposed attacks. The defense that harnesses this gap adjusts the acoustic information in the input audio to selectively allow only good inputs to be understood by machines while preventing bad ones from triggering actions on the target device, effectively causing those inputs to be discarded. Other proposed defenses against our proposed attacks notify, detect or prevent those attacks using a variety of tools and techniques, as described in Chapter 4.

1.5 Harnessing the Process of Audio Understanding in Humans

A domain that can potentially benefit from the way humans understand audio is Voice-over-IP (VoIP). The popularity of VoIP has significantly increased with the number of users growing from 57.4 million in 2012 to an estimated 72.2 million users by the end of 2019 [28]. Most VoIP systems use Variable Bitrate Encoding (VBR) as it conserves bandwidth to encode voice data before sending over the network. VoIP protocols often support end-to-end encryption of encoded voice data to prevent eavesdropping on VoIP communications. However, previous work has shown attacks on encrypted voice communications that allow an attacker to identify
spoken phrases and speakers’ identity without needing to decrypt VoIP communications [106, 158, 159, 160]. These attacks exploit the information leaked by packet sizes generated by the VBR codec to identify the speaker, gender of the speaker, language and phrases. Existing defenses against such attacks add different amounts of padding to encrypted packets to prevent information leakage due to packet sizes generated by the VBR codec [117, 161]. However, such defenses require support from both the sender and receiver in order to add extra padding on the sender side and strip that padding on the receiver side. This prohibits a privacy conscious participant in a VoIP communication to communicate with the other party who does not support these defenses in a privacy preserving manner.

This thesis proposes a technique that leverages the processes of audio understanding in humans to build better defenses against existing attacks on encrypted VoIP streams without requiring any participation from the receiver, as discussed in Chapter 5. At a high level, our proposed defense adds minimal extra audio to input frames before they are passed to VBR codec such that the sizes of output packets sent over the network do not leak any information about the contents of VoIP communication. The extra audio added to the underlying audio generated by the communicating user does not affect the audio as understood by the receiver due to the way the human brain processes audio. Thus, without requiring any support from the receiver, our approach leverages human understanding of audio to improve existing defenses.

1.6 ADDRESSING PRIVACY CHALLENGES

The popularity of personal voice assistants can be attributed to their high accuracy in correctly understanding users’ voice inputs. Improvements in speech recognition
algorithms and advances in hardware technology have enabled faster processing of machine learning algorithms, which has in turn made it possible for machines to transcribe audio in realtime with high accuracy.

To achieve high levels of accuracy, these voice recognition systems require a large corpora of spoken audio. The increasing popularity of voice interfaces that use cloud-backed speech recognition (e.g., Siri, Google Assistant, Amazon Alexa) increases the public’s vulnerability to voice synthesis attacks (explained below). For example, voice assistants like Alexa, Siri, Google Assistant and Cortana, record every voice input provided by the user and later use these recordings to improve their systems and services [2, 47, 48, 49]. Thus, the growing dependence on voice interfaces fosters the collection of our voices.

Entities with access to a person’s voice data may be able to craft fake voice samples and successfully impersonate that person. Voice synthesis uses a voice model to synthesize arbitrary phrases. Advances in voice synthesis have made it possible to create an accurate voice model of a targeted individual, which can then in turn be used to generate spoofed audio in his or her voice. Generating an accurate voice model of target’s voice requires the availability of a corpus of the target’s speech. Emerging products such as Adobe Voco [6], Lyrebird.ai [19, 38] and Google WaveNet [10] portray a disturbing picture for user privacy by promising to create fake audio as if spoken by a targeted person. This requires providing as little as one minute of speech audio of the targeted victim.

This thesis shows that voice recognition and voice accumulation (that is, the accumulation of users’ voices) are separable. We introduce techniques for locally sanitizing voice inputs before they are transmitted to the cloud for processing. Such methods employ audio processing techniques to remove distinctive voice character-
istics, leaving only the information that is necessary for the cloud-based services to perform speech recognition.

Our proposed technique leverages the process of audio understanding in machines that does not necessarily require very high quality audio input and can work with degraded audio quality up to a certain threshold. In essence, this is hidden voice commands, but applied as a defensive measure. We describe our approaches in Chapter 6.

1.7 Research Questions

This thesis explores the following research questions:

• **What are the practical implications of the differences between how humans and machines understand speech?** Humans and machines understand audio differently. Our work describes attacks and defenses that leverage this gap in audio understanding between humans and machines. Furthermore, we also identify other areas that can benefit from the way humans understand voice to improve security and performance.

• **Can the voice-input interface in devices be “stealthily” exploited?** Voice input is yet another interface accepting user input. The majority of devices accepting voice input operate in “always listening” mode and have no security mechanisms operating on the voice channel. This allows anyone to issue commands. We propose attacks against such devices that are harder for humans to detect and discuss the impacts of these attacks on security and user privacy.
• **How can the voice input interfaces on devices be secured against attacks?**
  Protecting voice input interfaces against unauthorized inputs is critical for protecting user privacy and ensuring user trust in the technology. This thesis describes and evaluates the feasibility and usability of various defenses to secure voice input interfaces.

• **Can we leverage the process of human understanding of audio to strengthen the security of existing systems?** In particular, we explore this question in the context of voice-over-IP systems.

• **Can voice data be scrubbed of identifying voice characteristics to minimize privacy exposure?** A person’s voice may be considered as his fingerprint. Voice data collected over time from a person may be used to forge audio in that person’s voice. We introduce techniques to sanitize voice data to reduce identifying characteristics of one’s voice while ensuring minimal impact on usability.

### 1.8 Contributions

This thesis makes the following contributions:

• **Stealthy attacks against voice recognition systems.** *(Chapter 3)* We propose novel attacks against voice assistants and speech recognition systems under black-box and white-box assumptions. These attacks use specially crafted audio command inputs that are understood correctly by a targeted speech recognition system but are not understandable by humans.
• **Defenses against attacks on voice recognition systems.** *(Chapter 4)* We describe various defenses against attacks against voice assistants and speech recognition systems and evaluate their advantages and disadvantages with respect to security and usability.

• **Improved defenses against re-identification attacks on encrypted VoIP streams.** *(Chapter 5)* By leveraging the process of audio understanding in humans, we propose improved defenses against known *re-identification attacks* on encrypted VoIP streams that do not require any support from receiver VoIP software.

• **Methods for more privacy preserving voice services.** *(Chapter 6)* Voice data collected by service providers from voice input contains much more information than is necessary just for voice recognition. This extra information could be used, for example, to generate false transcripts that appear to be from the speaker. We propose techniques to minimize the collection of identifying characteristics of an individual’s voice that have minimal impact on machine speech recognition.

## 1.9 Organization

The remainder of the thesis is organized as follows. Chapter 2 describes relevant prior work in speech recognition, attacks and defenses on VoIP communications and privacy of data. Our proposed attacks on voice command interfaces are described in Chapter 3. We present and evaluate defenses against the proposed attacks in Chapter 4. Chapter 5 describes proposed defenses against *re-identification attacks* on
encrypted VoIP streams. Emerging privacy threats from voice data collection and potential countermeasures are discussed in Chapter 6. We conclude in Chapter 7.
Chapter 2

Background and Related Work

This chapter provides relevant background knowledge for better understanding the work presented in the following chapters. We first briefly describe the process of speech recognition and various techniques that have been proposed and used to transcribe speech. The attacks proposed in this thesis target speech recognition, a machine learning system. We therefore provide an overview of other attacks that have been proposed against various machine learning systems in other domains and applications. This is followed by a review of prior work on attacks exploiting voice input interfaces and proposed defenses.

2.1 Speech Recognition Overview

The first step (pre-processing) in a typical speech recognition system usually involves removing background noise, filtering frequencies that are of little or no relevance for
speech recognition, and eliminating parts of the input signal that fall below an energy threshold. This process is generally referred to as speech/non-speech segmentation. A consequence of this segmentation process is that it generally disallows covert channels between the adversary and the device that cannot be perceived by the human operator. (An exception to this is the attack shown by Zhang et al. [169]) Conversely, it does not filter signals that have the requisite energy level and fall within the frequency range of human speech.

The filtered audio signal is then processed to extract acoustic features useful for recognizing speech (feature extraction). The input signal is converted from the time domain into the spectral domain by considering uniform length time frames and performing a Fast Fourier transform (FFT) over each frame. Most speech recognition systems use Mel-frequency cepstral coefficients (MFCC) to represent acoustic features of the input audio signal. MFCC closely approximates a human response to auditory sensation and allows for better representation of sound [88, 119].

The acoustic features extracted from the input signal are then matched against an existing model, built offline using training data. Speech recognition models are typically constructed using statistical approaches such as Hidden Markov Models [80, 100, 103, 109]. But with recent improvement in hardware capabilities, artificial neural networks have regained popularity for various machine learning tasks including automatic speech recognition [4, 55, 85, 110, 170]. These techniques are often used to map features to phonemes, and then to text. Based on the acoustic features of the input, the speech recognition model generates text predictions. A post-processing step may then be performed to rank the text predictions by employing additional sources of information—for example, enforcing grammar rules or considering the locality of words. Huang and Deng [99] describe a more thorough overview of the process of speech recognition.
2.2 Mel-frequency Cepstrum Coefficients

The attack we discuss in Chapter 3 leverages the common use of MFCCs to extract features from audio signals [88, 93, 101, 171]. In what follows, we present additional detail on MFCCs that may be useful to understand our attack.

Mel-frequency cepstral coefficients (MFCCs) are used to represent the short-term power spectrum of audio on a nonlinear mel frequency scale. MFCC is based on human hearing perceptions with frequency bands equally spaced on the mel scale (as compared to linearly spaced frequency bands in normal cepstrum) [119]. MFCCs are commonly derived by first taking the Fourier transform of a windowed excerpt of the input signal and mapping the powers of the spectrum obtained onto the mel scale. Next, log of powers for each frequency on the mel scale is taken followed by the discrete cosine transform of each mel log powers. The amplitudes of the resulting spectrum are the MFCCs.

Various parameters can be tuned for computing MFCCs: the window length to consider for computing the Fourier transform, the spacing between two windows, the number of warped spectral bands to use, the number of cepstral coefficients to return, and the lowest and highest band edge of the mel filters.

Our proposed attack involves tuning various parameters to compute MFCCs for unmodified input audio, and then reconstructing a modified audio signal from MFCCs. The MFCC parameters are tuned in a way that they (1) preserve enough audio features for the speech recognition system to correctly predict the corresponding text but (2) change the audio signal as perceived and understood by humans. We discuss this process in more detail in Chapter 3.
2.3 **Machine Learning Models for Speech Recognition**

Earlier speech recognition systems were built using statistic models such as Hidden Markov Models that relied on statistical properties of input audio [56, 69, 80, 100, 103, 109]. In contrast, more recent, state-of-the-art systems employ deep learning that use Deep Neural Networks (DNNs) and Recurrent Neural Networks (RNNs) with large number of hidden layers [70, 88, 93, 101, 171].

Statistical models for speech recognition typically rely on two components to map acoustic features extracted from input speech audio to the corresponding text: acoustic models and language models [56, 103, 130]. Acoustic models represent the relationship between the audio signal and the corresponding acoustic entities such as phonemes that make up speech. Knowledge about phonetics, speaker differences such as gender, dialect, context and environmental variations is captured by the acoustic model [99]. Language models represent the knowledge about a language's grammar and the likelihoods of sequences of words that make up valid sentences in that language [130]. Language models also provide additional information to match acoustic features with corresponding words and disambiguate between similar sounding words/phrases. Both acoustic and language models built using HMM use training data to capture the likelihood of given input acoustic features mapping to different sequence of words and the likelihood of given sequence of words occurring together respectively. We refer interested readers to existing work by Rabiner and Juang [130], Huang and Deng [99] and Deng and Li [69] for details of statistical speech recognition.

Statistical models such as HMM, however, have limitations due to assumptions about statistical properties. In particular, HMM-based speech recognition systems assume that there is no correlation between adjacent input audio frames, which is
not true for speech recognition tasks. Also, it is assumed that the probability of being in a given state only depends on the previous state. However, for speech recognition this may not true because dependencies for speech sounds can extend to several states \[104, 148\].

Neural networks, unlike statistical techniques, make fewer assumptions about statistical properties of features in the audio input. In parallel to statistical machine learning techniques for speech recognition task, the use of neural networks has been vastly explored \[88, 110, 132, 139, 140, 148, 155\]. Deep neural network-based approaches with multiple hidden layers have been shown to outperform statistical approaches such as HMMs \[84, 85, 170\]. Deep neural networks are trained using raw data and do not suffer from the artifacts of arbitrarily picking features used for training statistical models \[149\]. The majority of client facing, production speech recognition systems use some form of deep neural network based technique \[8, 9, 20, 82\]. These systems are usually trained as end-to-end speech recognition systems that learn different components required for speech recognition, e.g., acoustic and language models, simultaneously from raw audio data.

### 2.4 Attacks Against Machine Learning Systems

Attacks targeting machine learning systems, such as those used for speech recognition, have been widely studied under two broad categories: Evasion attacks and Poisoning attacks.

1. **Evasion Attacks.** Also known as “test-time” attacks, this category of attacks fools machine learning classifiers by providing manipulated input samples at test time. Depending on the application, the wrong classification can mean classifying illegitimate input as legitimate, evade detection, etc. It is assumed
that the attacker does not have access or the ability to manipulate the training
data or the model.

2. **Poisoning Attacks.** Also known as “training-time” attacks, this category of
attacks aims at compromising the machine learning classifier by manipulating
the training data. The attacker is assumed to have the ability to provide ma-
ligious training samples to the machine learning system and thus poison the
learned model.

Barreno et al. [58] provide an overview of various such attacks against machine learn-
ing systems. We briefly describe some examples of such attacks; a thorough survey
is beyond the scope of this thesis.

Machine learning systems used for spam filtering have been shown to be sus-
ceptible to evasion and poison attacks [111, 120, 121]. Lowd and Meek [111] show
an evasion attack against Naive Bayes and maximum entropy based spam filters. By
inserting or appending positive words to otherwise spam email, they are able to fool
the classifier into classifying spam messages as legitimate. Nelson et al. [120, 121]
describe poison attacks against SpamBayes that allows an adversary to attack the
spam filter to allow spam messages to pass through and also causing legitimate, non-
spam emails to be marked as spam. The attack modifies the machine learning model
parameters by sending specific content in emails to poison the model.

Work done by Xiao et al. [164] targets support vector machines (SVM) based
binary classifiers with poison attacks. By flipping the labels of training data that is
benign, the attacker can successfully compromise a binary SVM classifier through
label flips in training data. Biggio and Laskov [59] have further demonstrated poi-
soning attacks against SVM with non-linear kernels.
Machine learning based malware detectors have also been shown to be vulnerable to evasion attacks [86, 98, 144]. Srndic and Laskov [144] provide a case study of evasion attacks against PDFRate [1], an online malware detector that employs a random forest learning method to detect and score PDF malware samples. Assuming the knowledge of features used by the PDFRate system, their attacks succeed in lowering the classification scores for malware samples from 100% to about 33% by only modifying one-third of these features.

Recent work by Grosse et al. [86] targets neural network based malware classification by extending previous work in adversarial input generation in the visual domain [124] to construct adversarial malware samples. Their proposed technique generates adversarial attack samples that preserve malware’s functionality by only adding content to metadata files, but is able to fool a feed forward neural network based classifier, achieving 85% misclassification rate on the DREBIN Android malware dataset [53].

Hu and Tan [98] describe an algorithm to generate adversarial malware examples that can evade black-box machine learning based malware detectors. Their approach uses a Generative Adversarial Network (GAN) based algorithm to generate adversarial malware examples against detectors that use binary features. They also employ a substitute detector to fit the black-box malware detection system to extract gradient information that is used for generating adversarial malware samples. Extending this work, Hu and Tan [97] also demonstrate successful black-box attacks against RNN based malware classifiers by proposing a technique to generate sequential adversarial examples using a substitute RNN to approximate the targeted RNN based malware detector. While Rosenberg et al. [134] also describe ways to generate adversarial malware samples to fool RNN based malware detectors, their technique also allows for generating generic samples that can target other variants of RNN such
as LSTM, GRU, etc. as well as other traditional machine learning classifiers such as random forest and SVM, by using the principle of transferability in RNNs.

Attacks have also been proposed against machine learning systems that detect abusive language. Hosseini et al. [95] show an evasion attack to deceive Google’s Perspective API [35] built for detecting toxic phrases. They are able to attack the API by making simple manipulations to input text. By adding a dot between letters, misspelling a word, adding spaces between letters in a word etc., the system assigns lower toxicity scores to otherwise toxic phrases.

2.4.1 Attacks Against Machine Learning Systems in the Visual Domain

The attacks discussed above target machine learning systems used for applications where the human perception of test or training input is not observable. For example, manipulation of a malware sample to evade detectors cannot be directly observed by a human. On the contrary, in the visual domain, the human perception of an adversarial image aimed to fool an object recognition system is relevant. For a successful attack, the object in the image should not look like the output class label generated by the object recognition system. Since our work focuses on exploiting the differences between human and machine understanding of audio, we discuss similar work in visual domain.

Image and object recognition systems have been shown to be vulnerable to attacks where slight modifications to only a few pixels can change the resulting classification dramatically [83, 107, 122, 141, 146]. Szegedy et al. [146] first show that minor imperceptible perturbations to images, which would otherwise be correctly
classified by a deep neural network based classifier, cause these images to be misclassified by both the same as well as other models trained with different parameters. The same adversarial images can also be used against classifiers trained on disjoint training data set. Goodfellow et al. [83] provide an explanation to attacks proposed by Szegedy et al. [146], arguing that linear behavior in high-dimensional space opens up the feasibility of adversarial examples. These works assume white-box access to the machine learning classifier whereas our attacks (see Chapter 3) are not limited to speech recognition systems in which an attacker has white-box access; we also target speech recognition systems using just black-box access. Finally, Nguyen et al. [122] demonstrated attacks against object recognition systems using deep neural networks. They generate adversarial images using a gradient ascent method such that the generated images are unrecognizable to humans but are labeled as recognizable objects by the targeted system.

Attacks targeting biometric systems for authenticating humans have also been demonstrated [60, 141, 166]. Such attacks allow an attacker to pose as an authorized user by deceiving the system. Biggio et al. [60] show a poisoning attack against a face recognition system under the white-box model where the attacker knows how the target system works, what features are used, the matching algorithm, etc. The attack substitutes some of the training face templates with adversarial ones that are generated using their proposed methodology, allowing up to an 80% success rate. Xu et al. [166] demonstrate an evasion attack against facial recognition system by using public photos of targeted users to build live 3-D models. This allows the attacker to add motions to the virtual model, thus achieving realism to beat the liveness detectors used in facial recognition systems. Sharif et al. [141] have also demonstrated how automatic facial recognition systems can be attacked to impersonate another individual or simply fool the system to misclassify a person. Their attack requires
a human user to wear a printed pair of eyeglass frames. Although the attack succeeds in fooling the machine, it is not clear if human observers can also be deceived. On the contrary, our proposed attacks in the audio domain generate audio that is unintelligible to human listeners but is understood correctly by the targeted speech recognition system.

Recent work by Kurakin et al. [107] has shown that object recognition systems can also be fooled by using printed images of adversarial samples, thus, eliminating the need for direct modification of electronic image. Hosseini et al. [96] successfully fool Google's Cloud Vision API [33] by adding sufficient noise to the image, causing the API to generate completely different outputs for the noisy image.

In contrast to these attacks, our attacks work “over the air”; that is, we create audio that when played and recorded is recognized as speech. Recent attacks proposed by Evtimov et al. [78] target a deep neural network based classifier trained for detecting U.S. road signs by physical perturbations to physical “STOP” signs, causing them to be misclassified as “Speed Limit” signs with up to a 100% success rate.

Our attacks against voice recognition systems 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.

2.5 Attacks on Voice Interfaces

With the increasing popularity of voice interfaces, various attacks leveraging voice input as a vector have been proposed. Malicious applications have been shown to (1)
attack the voice channel for injecting commands [72] or (2) steal information by listening in on the audio voice channel [138]. Diao et al. [72] show that malicious apps on Android with zero permissions can exploit Google’s Voice Search. Their attack uses Android’s Intent mechanism to bring Google’s Voice Search to the foreground and then plays a pre-recorded audio via the device’s speaker in the background. The played audio is successfully recognized as a valid input by the voice search, thus allowing the attacker to inject and execute any arbitrary audio command. Similarly with the help of an installed malicious application on the smartphone, work by Schlegel et al. [138] propose a sound trojan called Soundcomber that can extract sensitive information from audio sensors on the smartphone. The attack requires an application with seemingly legitimate access to the device’s microphone to stealthily listen to audio communication with Interactive Voice Response (IVR) systems and extract sensitive information such as credit-card numbers, etc.

Although attacks that require malicious apps to be installed pose serious threats, such attacks are only limited to devices that have those malicious apps installed. On the contrary, our work focuses on directly targeting the voice channel without any need of malicious apps. Our attacks therefore, are not restricted to a particular device or user.

Attacks leveraging the accessibility features on various platforms have also exploited the voice channel. In particular, Jang et al. [102] have demonstrated that the accessibility features in modern operating systems can be used to bypass security checks and inject voice commands. For example, on the Windows operating system, their attack exploits the voice accessibility feature that runs with administrator privileges to execute arbitrary binaries. For the Android platform, it requires a malicious app to record the activation phrase to bypass the voice lock, perform a replay attack to enable voice input interface, and execute arbitrary voice commands by playing
the audio input through the device’s speakers. Such attacks, however, can be easily detected by users if they are in the vicinity of their device during the attack. Our work also executes unauthorized voice commands on devices; however, the voice commands are specifically crafted to be only recognizable by the targeted device and not by human listeners.

Most similar to our attacks is the work done by Esteves and Kasmi [77] on audio covert channels. Their proposed attack injects audio signals via headsets that must be plugged-in to act as an FM receiver antenna. By transmitting audio commands as FM signals via the plugged-in headphones, the device is fooled to believe that the input is coming from the microphone, thus, causing the voice interface to accept arbitrary audio input. However, the attack is limited to scenarios that require wired headphones with microphone to be plugged in. Additionally, the probability of detection of the attack is likely to be high as phone’s voice feedback will be relayed through the headphones to the user wearing them.

Following our work on attacks against voice assistants [65, 152], other researchers have also proposed similar attacks targeting voice assistants.

Abdullah et al. [52] employ audio manipulation in the time domain to create attack commands that are imperceptible to human listeners but are correctly understood by the targeted speech recognition system.

Another flavor of attacks proposed by various researchers [135, 136, 168, 169] renders the voice commands completely inaudible to human listeners. Such attacks modulate audible voice commands on ultrasound carrier waves to make them inaudible to human listeners. Once these ultrasound waves reach the targeted device, these attacks exploit the non-linearly of microphone circuitry to extract the modulated audible command and passes it to the speech recognition system. However,
these attacks do need to be tuned for each targeted device because of differences in
the microphone hardware and response to ultrasound waves.

2.6 Signal Pattern Matching

Researchers have also explored the use of signal features to identify individual de-
vices and keyboard entries. Zhuang et al. [172] predict the contents of typing from
the sound of keyboard clicks with the help of machine learning and speech recogni-
tion techniques. Dey et al. [71] identify individual smart devices by fingerprinting
accelerometer responses to motion stimulation, while Das et al. [68] achieve the
same goal by fingerprinting microphones and speakers embedded in smartphones.
Our work differs from these approaches in that we are essentially focusing on the
opposite—i.e., the manipulation of signal pattern matching to interpret seemingly
incomprehensible speech.

2.7 Defenses for Voice Interfaces

Previous research efforts on securing the voice input channel have largely focused
on controlling access to the microphone via access control mechanisms [21, 72, 91,
142, 167] and flow tracking and trusted paths [76, 102, 129, 133].

Diao et al. [72] propose modifying individual applications to prevent attacks
from untrusted input to the microphone by checking the status of other sensors, e.g.,
the phone’s speaker. However, such countermeasures are app specific and difficult
to deploy. Roesner et al. [133] propose user driven access controls for regulating
access to sensors and privacy-critical data by prompting users to decide whether the
requesting applications should be granted access to the requested resource or data.
Their technique derives the user’s access control intentions from the sequence of actions within the application. Mandatory access control (MAC) extensions proposed for the Android operating system [21, 142] allow for finer grained controls to regulate access to system resources such as the services of other applications, access to the network, microphone, camera, etc. Xu and Zhu [167] propose a privacy-aware sensor management framework, SemaDroid, for smartphones that allows users to manipulate the data generated by various sensors based on user-defined policies for the requesting application. Petracca et al. [129] implement an extension to SELinux, AuDroid, that tracks the creation of audio communication channels explicitly between applications and services that have access to both the microphone and speaker to prevent them from leaking sensitive information.

The attacks proposed in this thesis target legitimate apps that take voice as input and are actively used by the user. Our proposed attacks do not bypass any permissions or authorizations but simply obfuscates the audio input to prevent human understanding.
Chapter 3

Attacks Against Voice Interfaces

This chapter describes hidden voice commands, audio commands that are understandable by target speech recognition system but are considered noise by human listeners. The security implications of a “language” that can be understood by computer speech recognition engines but not by humans are somewhat subtle but important. Current digital assistants make no distinction between commands issued by their owners and those issued by unauthorized individuals. However, the user’s ability to hear spoken commands from others at least serves as a detection technique and allows the user to take mitigating action should others attempt to activate their device. When sounds that cannot be easily recognizable by humans as being speech are interpreted by a device, the opportunities for undetected unauthorized access to voice commands increase.

We exploit this opportunity—the gap between human and machine understanding of speech—to create hidden voice commands. This chapter describes how
hidden voice commands can be constructed. We detail general attack procedures for generating commands to target modern voice recognition systems.\textsuperscript{1}

3.1 \textbf{Attack Utility}

An adversary who is able to cause a target device to execute voice commands may leverage hidden voice commands to achieve the following goals: (this list is not intended to be exhaustive)

- \textit{Initiate a drive-by-download}: An attacker can issue commands to open a webpage that contains a drive-by-download. This effectively serves as a stepping stone, enabling other attacks that exploit vulnerabilities in the device’s browser;

- \textit{Earn money via pay-based SMS services}: An attacker can construct audio that causes phones to send text messages to pay-based SMS short code services that it operates;

- \textit{Enumerate devices in a physical area}: Similarly, an attacker may use a loudspeaker to cause nearby phones to send SMS messages to a number that the adversary controls, allowing it to enumerate the devices that are physically located within “earshot” of the broadcast (e.g., those belonging to dissidents attending a rally);

- \textit{Earn money via premium rate services}: An attacker can operate a premium rate number (i.e., a “900 number”) and monetize his attack by causing nearby phones to call it (for some mobile devices and calling plans);

\textsuperscript{1}We sometimes refer to these hidden voice commands as \textit{obfuscated commands}, in contrast to unmodified and understandable \textit{normal commands}. 

30
• Perform a denial-of-service attack: Using a public announcement system, an attacker can issue commands to turn on airplane mode on all devices, preventing them from receiving calls and other communications.

We note that the adversary may be required to issue multiple voice commands to activate the speech recognition system (“Hey Siri”) and launch his attack (“open myevilsite.com”). As more devices adopt voice activation and speech recognition capabilities, we anticipate that the security implications of our techniques will further increase. Simply put, any functionality that can be controlled via issuing voice commands is a potential target.

In what follows, we examine two classes of hidden voice commands. First, we explore black-box attacks in which we assume black-box access to the targeted speech recognition system by the attacker. The attacker does not know any details and internals such as parameters or machine learning algorithms used by the speech recognition system being targeted. Then, in § 3.3, we describe white-box attacks in which we assume that the attacker has full knowledge of the targeted speech recognition system.

3.2 BLACK-BOX ATTACK

3.2.1 THREAT MODEL AND ATTACKER ASSUMPTIONS

The black-box attack targets any device that is actively listening for voice input in an area where the attacker can introduce an audio signal. Many devices already continuously listen for voice activation commands: when plugged in, the iPhone responds to “Hey, Siri”, as does the Apple Watch whenever the wearer’s wrist is raised; many Android smartphones have the option of continuously listening for
“OK Google”; WearOS smart watches also listen for voice activation; certain GPS
devices such as Garmin’s Drive line respond to voice commands by default; and
in-home assistants such as Amazon Echo and Google Home actively listen for voice
commands. Given the rapid adoption of voice recognition systems it is reasonable to
assume that the trend of devices continuously listening for spoken activation phrases
will continue to expand in the future.

Under the black-box attack model, the adversary does not know the specifics
of algorithms, parameters, etc. used by the target speech recognition systems. We as-
sume 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 [101, 153], making our results widely
applicable.

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 guar-
anteed 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.

Corresponding to the vast majority of current-generation devices, our attacks
target devices that do not apply biometrics or otherwise attempt to authenticate the
speaker of the voice commands. We assume a human operator who is not using his
Figure 3.1: Adversary’s workflow for producing an obfuscated audio command from a normal command for black-box attack.

device at the time of the attack and therefore may not notice any on-screen notifications that reveal the adversary’s commands. Finally, we note that the attack may be targeted towards a particular device, or broadcast over a wide area to affect multiple devices.

3.2.2 Overview of Approach

Speech recognition systems rely on acoustic features extracted from input audio for generating text predictions. As long as the input audio contains enough acoustic information (above a certain threshold depending upon the sensitivity of the targeted system to noise, etc.), the system can correctly recognize the corresponding text with fairly high accuracy. A black-box attack modifies the input audio signal in such a way that the obfuscated audio output retains enough acoustic features for the speech recognition system to correctly predict the text while making it difficult for humans to understand the obfuscated audio signal.
Figure 3.1 shows the outline of the black-box attack. The attacker’s goal is to produce an obfuscated command that is accepted by the victim’s speech recognition system but is harder to understand 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 3.1, step 1) 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 2) 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 3) to the speech recognition system (step 4). 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 5).

If the candidate obfuscated audio is interpreted correctly (step 6), then the human attacker tests if it is human understandable. This step is clearly subjective and is subject to priming effects [116] 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 7). Otherwise, the candidate obfuscated audio command—which
Table 3.1: Mel-frequency cepstral coefficients parameters tuned to produce obfuscated audio.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wintime</td>
<td>time for which the signal is considered constant</td>
</tr>
<tr>
<td>hoptime</td>
<td>time step between adjacent windows</td>
</tr>
<tr>
<td>numcep</td>
<td>number of cepstral coefficients</td>
</tr>
<tr>
<td>nbands</td>
<td>no. of warped spectral bands for aggregating energy levels</td>
</tr>
</tbody>
</table>

is recognized by machines but not by humans—is used to conduct the actual attack (step 9).

3.2.2.1 MFCC Parameters

The MFCCs are coefficients that collectively make the MFC representation of sound. MFCC computation requires various parameters to be specified [75]. We focus on the four independent parameters shown in Table 3.1.

We experimentally observed the effect of changing each of these parameters independently on the perceived quality of obfuscated audio output. To check if the obfuscated audio can be understood by a speech recognition system, we submitted the outputs of the audio mangler to Google’s Speech Recognition engine using a publicly available API [126]. The API returns a list of possible transcriptions with a confidence value associated with each transcription. If the API cannot transcribe the audio, an error is returned.

Based on the edit distance between the transcriptions of normal and obfuscated audio signal, we manually narrow down the value range for each parameter. The range of parameter values, thus chosen, will produce obfuscated audio output with minimal but sufficient acoustic information for the targeted speech recognition system to transcribe the audio while making it non-trivial for a human listener to understand.
3.2.2.2 Feature Extraction with Tuned MFCC Parameters

After experimentally determining the range of MFCC parameter values, acoustic features are extracted by computing MFCCs [75] of the input signal using the chosen MFCC parameters. Computing MFCCs is lossy: the process considers the signal to be statistically constant over a small time window and also aggregates the energy levels of closely spaced frequencies to represent the total energy in various frequency regions on the mel-frequency scale. (The aggregation of energy level is motivated by the human hearing mechanism as it cannot discern differences between closely spaced frequencies and the effect becomes more dominant as the frequencies increase.) Thus, MFCCs do not retain all the information about the input audio signal. The tuned MFCC parameters used to create the attack command are intended to further increase this loss of information in a way that is detrimental to human understanding.

Recall that MFCCs are representations of acoustic features of an audio signal. We use the tuned MFCC parameters to compute MFCCs of an unmodified audio command. The resulting MFCCs contain just enough acoustic information, ensured by careful selection of MFCC parameters, such that an obfuscated audio signal reconstructed from them will be correctly recognized by the targeted speech recognition system.

3.2.2.3 Inverse MFCC

The extracted audio features represented as MFCCs are converted back to an audio signal by reversing the steps of the MFCC computation. The inversion of MFCCs back to a waveform involves the addition of noise, since MFCC computation is lossy and aggregates energy levels of closely spaced frequencies into frequency regions.
Inversion from MFCCs to audio signals *mangles* the original audio signals, making them difficult for human listeners to understand.

In summary, our approach modifies the input signal by (1) adjusting MFCC parameters and performing feature extraction, and then (2) reconstructing an audio signal by applying a reverse MFCC to the extracted features. A key property of our approach is that the features used in the speech recognition system remain mostly undisturbed (since they are extracted and then reconstructed), while the non-extracted-features are lost in the reconstruction.

It is worth stressing that the adversary can perform the mangling entirely off-line. Recall that the adversary’s job is to construct an audio file that is interpreted by computer speech recognition systems, but not easily discernible to humans. He can thus perform the procedure in a trial-and-error fashion, tuning the audio mangler’s parameters to generate audio files and testing them manually to see whether the obfuscated output is accepted by a copy of the targeted speech recognition system and to see if the a human listener deems the obfuscated audio to be non-understandable. This process can be time-consuming, but producing a *single* obfuscated audio that achieves the above two constraints will likely lead to a successful attack.

### 3.2.3 Evaluation of Black-Box Attacks

The goal of our black-box attack is to produce obfuscated audio files that activate commands on voice-recognition systems but are difficult for humans to understand. We evaluate the effectiveness of this attack by implementing a set of obfuscated commands, verifying that they do activate the phone, and then performing human testing to determine how difficult the commands were for humans to interpret.
3.2.3.1 **Experimental Setup**

We test our proposed black-box attacks by tuning four MFC parameters to mangle and test audio using the workflow described in § 3.2.2 to determine the ranges for human and machine perception of voice commands.

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 ($P_{\text{noise}}^{\text{dB}}$) of approximately 53 dB. (A previous version of the attack employed other commands as well [152].)

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 default speech recognition system was used to interpret the commands [4]. 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 ($P_{\text{signal}}^{\text{dB}}$) to be approximately 88 dB.

We also projected various background noise samples collected from SoundBible [45], 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
Table 3.2: Black-box attack results.

<table>
<thead>
<tr>
<th></th>
<th>Ok Google</th>
<th>Turn on airplane mode</th>
<th>Call 911</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Machine</td>
<td>Human</td>
<td>Machine</td>
</tr>
<tr>
<td>Normal</td>
<td>90% (36/40)</td>
<td>89% (356/400)</td>
<td>75% (30/40)</td>
</tr>
<tr>
<td>Obfuscated</td>
<td>95% (38/40)</td>
<td>22% (86/376)</td>
<td>45% (18/40)</td>
</tr>
</tbody>
</table>

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, we assessed whether the Turk workers correctly understood the commands.

devices and recorded all audio commands that we played to the devices for use in later experiments, described below.

3.2.3.2 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.
Figure 3.2: 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.

3.2.3.3 Machine Understanding

Table 3.2 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 3.2 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. Detailed results for additional background noises appear in our USENIX Security paper [65].

3.2.3.4 Human Understanding

To test human understanding of the obfuscated voice commands, we conducted a study on Amazon Mechanical Turk\(^2\), a service that pays 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

\(^2\)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 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”).
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 3.2 under the “human” columns.

We additionally required that workers be over 18 years of age, citizens of the United States, and not work for Georgetown University. 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.

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 3.2 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.

3.2.3.5 Objective Measures of Human Understanding

The analysis above is based on our assessment of Turk workers’ transcripts. 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
Table 3.3: Percentages of human listeners who were able to correctly comprehend at least 50% of the black-box attack voice commands.

<table>
<thead>
<tr>
<th></th>
<th>OK Google</th>
<th>Turn On Airplane Mode</th>
<th>Call 911</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Normal</strong></td>
<td>97% (97/100)</td>
<td>89% (102/114)</td>
<td>92% (75/81)</td>
</tr>
<tr>
<td><strong>Obfuscated</strong></td>
<td>24% (23/94)</td>
<td>47% (52/111)</td>
<td>95% (62/65)</td>
</tr>
</tbody>
</table>

slightly different—whether a human is able to reconstruct the sounds of an obfuscated command—and does not directly capture understanding.

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.

To verify our evaluation 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 $\delta$ as the Levenshtein edit distance between phonemes of two transcriptions. We define $\phi$ as $\delta/L$, where $L$ is the phoneme length of normal command sentence. The use of $\phi$ reflects how close the transcriptions might sound to a human listener. $\phi < 0.5$ indicates that the human listener successfully 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 3.3 shows the results of our binary classification. The difference in success rates of normal and obfuscated commands is similar to that of human listeners in Table 3.2, 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 $\delta/(\delta + L)$, where $L$ is again the phoneme length of normal command sentence. The relative phoneme edit distance has a range of
Figure 3.3: Cumulative distribution of relative phoneme edit distances of Amazon Mechanical Turk workers’ transcriptions for “OK Google”, “Turn on Airplane Mode” and “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.

[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 3.3 shows the CDF of the relative phoneme edit distance for the (top) “OK Google”, (middle) “Turn on Airplane Mode” and (bottom) “Call 911” voice commands. These graphs show similar results as reported in Table 3.2: Turk workers were adept at correctly transcribing normal commands even in presence of 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 obfuscated commands: less than 30% were able to achieve a relative edit distance less than 0.2 for “OK Google” and “Turn on Airplane Mode”.

3.3 **White-Box Attack**

The white-box attack considers an attacker who has knowledge of the underlying speech recognition system. To demonstrate this attack, we construct hidden voice commands that are accepted by the open-source CMU Sphinx speech recognition system \[108\]. CMU Sphinx is used for speech recognition by a number of apps and platforms\(^3\), making it likely that these white-box attacks are also practical against these applications. We note that the work presented in the following section was collaboratively done with the co-authors of our USENIX Security paper \[65\]; the co-authors proposed the white-box attack. We present a brief overview of white-box attack and the results for coherence in this thesis. Interested readers may refer to the USENIX Security paper \[65\] for further details.

3.3.1 **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.

\(^3\) 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](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.

3.3.1.1 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 our USENIX Security paper [65].

3.3.1.2 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.

### 3.3.2 Attacker Assumptions

We assume that the attacker has complete knowledge of the target speech recognition system and can interact with it at will while creating an attack. We also assume that the attacker knows the parameters used by the speech recognition algorithm.\(^4\)

### 3.3.3 Overview of Approach

Figure 3.4 shows the high level idea used for generating white-box attack commands. White-box attack commands are generated by going through the speech recognition pipeline in reverse. That is, the command generation process takes the desired command text as the input and outputs the attack audio.

Going in the reverse direction of a speech recognition pipeline is made possible by the attacker’s knowledge of its internals, including the machine learning algorithm and parameters. Thus, in contrast to the black-box attack model in which

\(^4\)Papernot et al. [123] demonstrated that it is often possible to transform a white-box attack into a black-box attack by using the black-box as an oracle and reconstructing the model and using the reconstructed parameters.
the attacker starts from an actual audio, the attacker first chooses the desired command text as the first step under the white-box model. The attacker then tries to find a set of feature vectors which, when supplied as an input to the machine learning model of the speech recognition system, produces the required transcription text. Once the corresponding set of feature vectors to the desired transcription text is found, the attacker finds an input audio signal, which when processed by the feature extraction step, produces the feature vectors close to those corresponding to the desired transcription text.

3.3.4 White-Box Attack Summary

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.

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.

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. We consider one frame at a time and use the least squares approach to find the next best frame using gradient descent.

Interested readers may refer to our USENIX Security paper [65] for further details of white-box attack generation.

3.3.5 Evaluation of White-Box Attack

3.3.5.1 Machine Understanding

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 our USENIX Security paper [65]. 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.

### 3.3.5.2 Human Understanding

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.

We 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 3.4 lists the results, showing that normal commands were
Table 3.4: White-box attack results.

<table>
<thead>
<tr>
<th></th>
<th>Human Understanding</th>
<th>Machine Understanding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>74% (230/310)</td>
<td>–</td>
</tr>
<tr>
<td>Obfuscated</td>
<td>0% (1/377)</td>
<td>82% (82/100)</td>
</tr>
</tbody>
</table>

Percentages show successful comprehension of Normal and Obfuscated version of voice commands for humans and machines. Our white-box attack constructs the obfuscated attack without a starting normal attack; we only evaluate the machine understanding of the obfuscated commands.

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. While 97% of human listeners were able to comprehend at least 50% of phonemes for normal commands, but only 10% were able to do so for obfuscated commands generated using white-box attack technique.

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.
3.4 Summary

While ubiquitous voice-recognition brings many benefits, its security implications are not well studied. This chapter investigates 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 and more difficult-to-detect attacks are possible.
Chapter 4

Defenses Against Hidden Voice Commands

The majority of voice assistants today lack robust security mechanisms to defend against unauthorized voice commands, including the hidden voice commands proposed in Chapter 3. Devices that provide some level of protections against unauthorized use of voice input either require physical interaction with the device for authentication, e.g., unlocking the phone, or rely on ad hoc mechanisms to prevent execution of certain voice commands, e.g., Amazon Echo has the option that requires a 4-digit PIN for making purchases over the voice channel. However, such defenses either inhibit user experience over the voice channel or lack scalability.

In this chapter, we explore potential defenses for hidden voice commands 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.
4.1 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).

4.1.1 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 that 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 beeps and begin to ignore them. 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 required 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 over-simplifies 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 our USENIX Security paper [65].
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 [137, 145, 162]. 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.

4.2 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 affecting 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 can 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.

4.2.1 The Audio CAPTCHA

Such a confirmation system already exists in the form of audio CAPTCHAs [87]. An audio CAPTCHA is a challenge-response protocol in which the challenge consists of speech that is constructed to be difficult for computers to recognize while being relatively 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 a 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 [62, 147]. As audio CAPTCHAs have improved over time [67, 115], 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 [44] offers audio challenges initially consisting of five random digits spread over approximately ten seconds; and NLP Captcha [42] provides audio challenges
of about three seconds each composed of four or five alphanumeric characters, with the addition of the word “and” before the last character in some challenges.

We tested 50 challenges each of reCaptcha and NLP Captcha in October, 2016 by segmenting the audio challenges before transcribing them using Google’s speech recognition service. Figure 4.1 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 seem 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.

4.3 Defenses that Detect and Prevent

4.3.1 Speaker recognition

Speaker recognition (sometimes called voice authentication) has been well-explored as a biometric for authentication [64]. Google now includes speaker recognition as
an optional feature in its Android platform [29]. Apple has also introduced similar functionality in iOS [36]. To provide a more personalized experience, Google home now supports speaker recognition on the wakeup “Ok Google” phrase to differentiate between different users [34] but does not offer continuous speaker authentication.

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\(^1\); for example, Aylett and Yamagishi [54] are

\(^1\)CereVoice offers an online service for “[creating] a computer version of your own voice” [30].
able to mimic President George W. Bush’s voice with as little of 10 minutes of his speech. Upcoming speech synthesis services such as Lyrebird.ai [38] claim to build a voice model of a person’s voice with only one minute 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 disable 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 unsuitable 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.

4.3.2 The “Filter”

The “Filter” 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) [118]. The LDC corpus consists of American English-language speakers, 30 of them speaking the 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 §3.2. 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 $\delta$ 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 4.2 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
Figure 4.2: The cumulative distribution of relative phoneme edit distances for normal and obfuscated commands, for filtering rates of 80% and 92%.

unacceptable for users who depend on voice functionality. Through 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%.

4.3.3 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
Table 4.1: Confusion matrix of our classifier.

<table>
<thead>
<tr>
<th></th>
<th>Normal</th>
<th>Obfuscated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>49.9%</td>
<td>0.1%</td>
</tr>
<tr>
<td>Obfuscated</td>
<td>0.1%</td>
<td>49.9%</td>
</tr>
</tbody>
</table>

Table 4.2: Confusion matrix of our classifier, when classifying audio from outside corpora.

<table>
<thead>
<tr>
<th></th>
<th>Normal</th>
<th>Attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>White-box Attack</td>
<td>30.7%</td>
<td>69.3%</td>
</tr>
<tr>
<td>Accent GMU</td>
<td>99.2%</td>
<td>0.8%</td>
</tr>
</tbody>
</table>

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 [65], extracted using the pyAudioAnalysis library [81] 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 § 3.2, we produce obfuscated hidden voice commands, which we label as obfuscated. We overlay the background noise samples described in § 3.2.3.1 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 4.1. 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 Chapter 3 and audio from the Accent GMU dataset [157]. 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 4.2. 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 backend speech recognition system, the classifier is able to flag nearly 70% of the hidden voice commands as being malicious.

4.4 Summary of Defenses

We present the first examination of defenses against hidden voice commands. Our analysis of notification defenses (§ 4.1) 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 (§ 4.2) 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 (§ 4.3).
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.

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.
Chapter 5

**Whisper: A Novel Defense Against VoIP Traffic Re-Identification Attacks**

Encrypted voice-over-IP (VoIP) communication often uses variable bit rate (VBR) codecs to achieve good audio quality while minimizing bandwidth costs. Prior work has shown that encrypted VBR-based VoIP streams are vulnerable to re-identification attacks in which an attacker can infer attributes (e.g., the language being spoken, the identities of the speakers, and key phrases) about the underlying audio by analyzing the distribution of packet sizes. Existing defenses require the participation of both the sender and receiver to secure their VoIP communications.

This chapter presents *Whisper*, the first unilateral defense against re-identification attacks on encrypted VoIP streams. Whisper works by modifying the audio signal before it is encoded by the VBR codec, adding inaudible audio that either falls outside the fixed range of human hearing or is within the human
audible range but is nearly imperceptible due to its low amplitude. By carefully inserting such noise, Whisper modifies the audio stream’s distribution of packet sizes, significantly decreasing the accuracy of re-identification attacks. Its use is imperceptible by the (human) receiver.

Whisper can be instrumented as an audio driver and requires no changes to existing (potentially closed-source) VoIP software. Since it is a unilateral defense, it can be applied at will by a user to enhance the privacy of its voice communications. We demonstrate that Whisper significantly reduces the accuracy of re-identification attacks and incurs only a small degradation in audio quality.

5.1 Introduction

Voice-over-IP (VoIP) systems encode voice for transmission over a network. The majority of popular VoIP systems use variable bitrate encoding (VBR) to achieve high quality audio while conserving bandwidth. The output data of VBR per unit time depends on the complexity of the input audio, resulting in audio frames (and ultimately, packets) of various sizes. To secure encoded voice data during transmission, VoIP systems often support end-to-end encryption, for example, via secure real-time transport protocol (SRTP).

Such protocols, however, while ensuring message confidentiality, still leak information about the underlying audio. Prior work has shown that significant information about the audio stream—including the identity of the speaker, the gender of the speaker, the spoken language, and even key phrases—can be inferred by analyzing the distribution of encrypted packet sizes [106, 117, 158, 159, 160]. Such re-identification attacks are possible because the size of the encrypted packets de-
Figure 5.1: Overview of a traffic re-identification attack on an encrypted VoIP stream.

Typically, re-identification attacks use machine learning techniques to infer information about the encoded audio from the encrypted VoIP stream. Figure 5.1 shows an overview of the attack’s workflow. An adversary intercepts the encrypted VoIP stream and extracts features from the distribution of encrypted packet sizes. Using a labeled training corpus, the adversary applies machine learning techniques to build a classifier (again, using features based on the distribution of encrypted packet sizes), and applies the classifier to extract information about the underlying audio. Modern encrypted VoIP systems are surprisingly vulnerable to such attacks; for example, Wright et al. [159] showed that the spoken language can be inferred with 87% accuracy when presented as a binary classification problem and 66% accuracy using a 21-way (i.e., 21-language) classifier.

A straightforward and effective defense against re-identification attacks is to abandon VBR in favor of constant bitrate encoding (CBR). CBR offers complete protection against re-identification attacks since it eliminates information leakage due to packet size. However, for the same targeted bitrate, VBR offers far better
audio than CBR. Reliance on CBR incurs such a bandwidth overhead that we know of no encrypted VoIP system that has opted to use it.

There are also existing defenses that attempt to disrupt re-identification attacks while still permitting the use of VBR codecs [117, 161]. These function by modifying the size of packets generated by a VBR codec, thus hiding the underlying packet size distribution of the encoded audio. However, existing defenses either require participation of both the sender and receiver [117, 161] or require white-box access to the VBR codec [161].

In this chapter, we propose Whisper, a unilateral defense against re-identification attacks. Whisper leverages the limits of the human audible range to alter the size of packets generated by a VBR codec in a manner that (i) obfuscates the true packet size distribution and (ii) is (ideally) imperceptible to the receiver. It allows a privacy conscious sender to secure his side of the communication without any support from the receiver.
Whisper alters the size of output packets generated by a VBR codec by overlaying tuning audio on the actual audio before encoding occurs. The addition of tuning audio changes the characteristics of the original audio signal to be encoded, without affecting the contents of the original audio as perceived by the human listener on the receiver side.

At first blush, it may seem that Whisper is inherently incompatible with modern audio codecs, since codecs often use band filters to remove audio outside of the human audible range. However, in practice, codecs typically err on the side of preserving audio quality and are inexact in their filtering. This leads to segments of the spectrum that are both not-filtered and either inaudible or unplayable due to the limits of commodity speakers.

Figure 5.2 shows an overview of Whisper. Whisper overlays tuning audio to the input audio frame before it is encoded by the VBR codec in the VoIP application. It is thus agnostic to the particular VoIP application, which we assume is unaware of the Whisper protections and merely receives audio data from a Whisper-enabled audio/microphone driver. In summary, Whisper’s unilateral protections and ability to be used with any closed-source (i.e., black-box) VoIP software enable the practical protection of communication using deployed VoIP systems.

We evaluate Whisper using large voice corpora and the popular Opus VBR codec [43]. We show that Whisper significantly reduces the accuracy of re-identification attacks. For example, Whisper decreases the adversary’s accuracy to correctly identify the speaker of an encoded VoIP conversation from 97.22% (without Whisper) to 31.13% (with Whisper). Whisper incurs limited bandwidth overhead and has no significant impact on the quality of actual audio.
5.2 Background on VoIP Attacks and Defenses

VoIP re-identification attacks are instances of traffic fingerprinting, the latter of which has been richly explored (see, for example, early work by Hintz [94] and Crotti et al. [66]). Traffic fingerprinting attempts to infer characteristics about communication by examining its network attributes (e.g., the timing, sizes, and inter-arrival times of packets; and their distributions) rather than by analyzing the communication’s contents. There is an active arms race between website traffic fingerprinting techniques and defenses [63, 156], which is especially relevant to anonymity networks such as Tor [73].

VoIP re-identification attacks apply similar fingerprinting techniques to identify attributes of the underlying call audio and/or the participants of the communication. Prior studies have found the distribution of (encrypted) packet sizes to be sufficient to infer with high accuracy the language being spoken [159], the gender and identity of the speaker [117], and even key phrases [158, 160].

Wright et al. [161] first proposed a defense against statistical traffic analysis of VoIP streams by morphing one class of traffic to look like another class. Their proposed defense alters the packet sizes of the source traffic such that the statistical distribution of its encoded packet sizes closely matches that of the target traffic. With only black-box knowledge of the codec, their defense increases the packet sizes by padding the encoded output of a VBR codec. With white-box access to the codec, Wright et al. [161] rely on the selection of the bit rate within the codec to increase or decrease the size of an encoded packet. To find a distribution closest to that of the target traffic, Wright et al. use comparison functions such as the $\chi^2$ statistic and convex optimization to minimize the overhead due to the padding of packets.
Rather than morphing the source distribution to a particular target distribution, Moore et al. [117] calculate a new, synthetic, “superdistribution” to which all source traffic distributions can be morphed. To calculate the superdistribution, their defense considers the distributions of all potential source traffic and determines the least bandwidth-intensive distribution that can be used to map all of the source traffic. Once the superdistribution has been determined, the output of the VBR codec is padded to map it to the size described by the superdistribution. Because the padding is itself encrypted end-to-end, an attacker cannot easily infer the original, unpadded distribution of packet sizes. Whisper’s approach to determine how much noise/padding to add to the baseline traffic borrows from Moore et al.’s algorithm.

5.2.1 LIMITATIONS OF EXISTING DEFENSES

A straightforward defense to prevent the leakage of information due to traffic analysis of packet sizes is to use constant bitrate encoding (CBR). However, to achieve the same audio quality as VBR, CBR incurs significant bandwidth overheads. This makes CBR unsuitable for networks with limited bandwidth such as cellular networks.

The major limitation of existing traffic morphing defenses for VoIP streams [117, 161] is that they require both communicating parties to support and participate in the defenses. Defenses proposed by both Wright et al. [161] and Moore et al. [117] add padding to alter the size of the encrypted packets on the sender side, requiring the receiver to strip the extra padding. In cases where the receiver does not support the removal of the extra padding, the sender can only communicate over the vulnerable VBR channel. This essentially prohibits a privacy conscious participant from communicating with another party who does not support these defenses.
In contrast, our proposed Whisper defense is unilateral and does not require the participation of the receiver. Currently, to our knowledge, no deployed VoIP system supports a unilateral defense that can prevent traffic analysis of encrypted VoIP streams while supporting VBR encoding. Our techniques can be implemented as a virtual device driver, and are therefore compatible with existing closed-source VoIP software (e.g., Skype).

Additionally, the approach taken by Wright et al. [161] of changing the codec’s bitrate to manipulate packet sizes requires white-box access to the VBR codec. In contrast, Whisper takes a black-box approach and can work with applications that do not allow access to the codec or its settings.

5.3 User and Attack Models

We assume two parties communicating via a VoIP application that uses VBR encoding. The VoIP application provides end-to-end encryption; that is, it encrypts all traffic between the communicating parties to prevent eavesdropping, but does not make any effort to hide the size of the encrypted packets. Furthermore, we assume that the communicating parties use a closed-source VoIP application such as Skype and are unable to modify the codec parameters. This assumption enforces the constraint that the defense should work with popular VoIP clients without requiring any modifications to them.

As with previous work [117], Whisper assumes that the VBR codec used by the VoIP application is publicly known. Whisper requires some per-codec tuning, which necessitates having black box access to the codec. This is a realistic assumption since popular VoIP clients use standardized codecs whose implementations are publicly available. For example, Skype uses the Silk codec [154] while WhatsApp is
known to use the Opus codec [43, 89, 131], both of which have publicly available implementations. We do not require that the codec is itself open source; rather, we require only that an implementation is available for tuning our defense.

Since VoIP is typically a bidirectional channel, it should be emphasized that Whisper protects only the communication that is generated by the party applying Whisper. We do not consider correlation attacks in which the unprotected direction is used to infer information about the channel being protected by Whisper; this is likely feasible for inferring language (since typically both communicants use the same language), but may be difficult for re-identification attacks that attempt to perform speaker identification or identify key phrases. Of course, Whisper can be used by both parties to provide bidirectional protections.

Our attacker model follows existing work [117, 161] with respect to the adversary’s capabilities and access to training data for performing traffic analysis. We consider a passive adversary that intercepts all encrypted VoIP traffic between the communicating parties. The adversary does not have access to the underlying plaintext audio. However, it can inspect the traffic and learn other characteristics, including the size and timing of packets.

The adversary’s goal is to use the distribution of packet sizes obtained from the encrypted packet stream to discern information about the underlying audio. In this chapter, we focus on the case in which the adversary attempts to learn the identity of the speaker, given a closed-world setting in which the set of candidate speakers is known apriori. We emphasize that the closed-world setting is a conservative model (for the defense). That is, a defense that successfully thwarts accurate re-identification in the (worse-case) closed-world setting is also effective in the open-world setting in which all speakers (or languages, genders, phrases, etc.) must be considered.
We chose to consider speaker identification—as opposed to re-identifying gender or language—as speaker identification has been previously shown to be highly accurate [117] and arguably more interesting to potential eavesdroppers than gender or language identification. (Presumably, learning the identity of the speaker also provides hints at gender and language.)

To conduct its attack, the adversary has access to a training corpus of unencrypted audio samples, including samples from all potential speakers in our closed-world setting. The adversary also has complete knowledge of the Whisper algorithm and its parameters, excluding the private random bits generated by the sender.

As shown in Figure 5.1, the adversary uses the training corpus to build a machine learning classifier to learn information about the encoded audio from the encrypted packet size distribution. All known re-identification attacks on encrypted VoIP streams [106, 117, 158, 159, 160] consider the frequency of $n$-grams over the size of packets as features to the machine learning classifier. We use a similar approach to show the vulnerability of the Opus codec [43] to re-identification attacks and to evaluate the effectiveness of Whisper in mitigating such attacks.

5.4 Methodology

Re-identification attacks on encrypted VoIP streams leverage the packet size distribution of the encrypted VoIP packets to perform traffic analysis. Whisper defeats such attacks by changing the size of the encrypted packets generated by the VoIP application before they are sent over the network. The updated sizes of these encrypted packets should be such that their packet size distribution decreases the information leaked by the encrypted VoIP stream and reduces the ability of the adversary to perform accurate traffic analysis.
Moore et al. [117] propose padding packets to achieve a particular distribution—the superdistribution—to which all classes of traffic (e.g., different speakers, genders, phrases, etc.) can be mapped. Conceptually, morphing all underlying (and revealing) distributions to the superdistribution hinders re-identification attacks since it removes the adversary’s ability to discover distinguishing features within the packet size distribution.

Whisper borrows the superdistribution concept from Moore et al. [117], but uses inaudible audio to enable unilateral protections. In what follows, we provide a brief overview of the superdistribution generation (§5.4.1) and mapping techniques (§5.4.2), and then describe how Whisper uses inaudible noise to morph traffic to the superdistribution (§5.4.3).

### 5.4.1 Creating the Superdistribution

Moore et al. [117] construct a superdistribution using an audio corpus, which we will refer to as the training corpus. (They conservatively assume that the adversary also has access to this corpus.) Without loss of generality, we will describe both the defense of Moore et al. and our Whisper system in terms of defending against re-identification attacks that aim to identify a speaker from a closed set of potential speakers. Our defenses are equally applicable to other re-identification tasks.

We assume a VBR codec that produces a sequence (vector) of \( L \) audio frames \( \vec{a} = \langle a_1, \ldots, a_L \rangle \), where each audio frame encodes a fixed-length time period of the audio (usually 20 ms) and \( L \) is a function of the length of the source audio. That is, \( \vec{a} \) is the encoding of the input audio sample produced by the VBR encoder. We consider the set of possible packet sizes over \( \vec{a} \) to be the codec alphabet (\( \Sigma_{in} \)) of that codec. We note that \( \Sigma_{in} \) is finite, and treat it as an ordered set \( \Sigma_{in} = \{ \Sigma_1, \Sigma_2, \ldots, \Sigma_{|\Sigma_{in}|} \} \) where \( \Sigma_i < \Sigma_j \) when \( i < j \), for all \( i, j \in [1, |\Sigma_{in}|], i \neq j \).
The superdistribution generation algorithm considers the distribution of all
the speakers in the training corpus and calculates the least bandwidth-intensive dis-
tribution that can be used as the target distribution. To preserve audio quality, we
are limited to additive modifications only: we can pad any audio sample \( a_q \in \tilde{a} \) of
size \( \Sigma_i \) to any size larger than \( \Sigma_i \), but cannot decrease the size of \( a_q \) without signifi-
cantly degrading audio quality. While the defense of Moore et al. does not require
that the set of padded packet sizes (\( \Sigma_{out} \)) equal that of the codec (i.e., \( \Sigma_{in} \)), Whisper
necessitates that \( \Sigma_{out} = \Sigma_{in} \) since the receiver should be agnostic (and potentially
unaware) of the defense’s use. For clarity, in what follows, we assume that \( \Sigma_{out} = \Sigma_{in} \)
and use \( \Sigma \) as shorthand.

We directly apply the superdistribution generation algorithm of Moore et al.
[117, see Algorithm 1]. Briefly, the superdistribution generation algorithm consid-
ers the packet size distributions for each speaker in the training corpus, and then
calculates the least bandwidth-intensive distribution to which the packet size distri-
bution of all the speakers in the training corpus can be morphed. For the ascending
list \( L_z = \langle l_{1z}, \cdots, l_{kz} \rangle \) of \( k \) different possible lengths of output packet sizes for a
packet stream \( z \), the superdistribution algorithm calculates a target distribution \( L_t \)
such that for all \( 1 \leq i \leq k, \sum (l_{it} + \cdots + l_{kt}) = \max_z (\sum (l_{iz} + \cdots + l_{kz})) \) over all packet
streams \( z \) in the training corpus.

We assume an adversary will consider not just the relative frequency of packet
sizes, but also the relative frequency of \( n \)-grams of packet sizes. (This is the predomi-
nant approach used by prior work on re-identification attacks [117, 158, 159, 160].)
More precisely, the adversary uses overlapping sequences of length \( n \) over the output
packet sizes in \( \Sigma \) as features for the machine learning classifier. Like the approach
of Moore et al. [117], Whisper’s superdistribution generation algorithm computes a
separate superdistribution for each unique sequence of \( n - 1 \) packet lengths. Thus,
there will be $|\Sigma|^{n-1}$ superdistributions. During the mapping step (see §5.4.2), the superdistribution matching the last $n-1$ packet length sequence is used to determine the target packet size of the next packet to be encoded.

### 5.4.2 Mapping to the Superdistribution

Whisper uses the superdistribution to determine the desired size of the next outgoing packet from the VoIP application. That is, given an input audio frame $a_q$ of size $\Sigma_i$ and the history of previously transmitted audio frames (including their added noise), Whisper determines the desired augmented packet size $\Sigma'_i$ (where $\Sigma'_i \geq \Sigma_i$) that will cause the distribution of packet sizes to appear closest to that of the superdistribution. (As discussed above, $\Sigma'_i$ cannot be less than $\Sigma_i$ without incurring a significant loss of audio quality.)

We make a slight modification to the mapping algorithm proposed by Moore et al. [117] to include additional parameters to allow a trade-off between security and bandwidth overhead. (We use the term *packets* to be consistent with the terminology of Moore et al. [117]. Moore et al.’s defense added padding to the encoded packets generated by the VoIP application. In contrast, Whisper modifies audio frames before they are encoded by the VoIP application.)

Algorithm 1 describes how Whisper calculates the target packet size from an input stream such that the distribution of target packet sizes closely resembles the superdistribution. The mapping algorithm works for any level of $n$-grams.

In lines 5-8, we pad the initial $n-1$ packets (audio frames) to the maximum packet size for bootstrapping. Next, for each input audio frame, line 11 computes the cost of choosing each possible packet size based on the current distribution of the last $n-1$ output packet sizes and the target distribution.
Algorithm 1 Mapping an input distribution to output distribution determined by the superdistribution.

1: procedure MorphStream(inputStream, targetStream, numPktSizes, NgramSize, pktSizeWeights, strictness)
2:  
3: currentDistCounts ← Empty array of size numPktSizes × NgramSize
4: lastNPkts ← Empty queue of packet sizes
5: maxSizePkt ← Size of largest packet in inputStream
6: for x in range(0, NgramSize) do
7:  currentPkt ← inputStream.dequeue()
8:  doWhisper(currentPkt, maxSizePkt)
9:  lastNPkts.enqueue(currentPkt.size())
10: while currentPkt ← inputStream.dequeue() do
11:  ▷ Cost of choosing each potential target packet size.
12:  sizeCosts ← computeSizeCosts(currentPkt.size(), targetStream, maxSizePkt, currentDistCounts, lastNPkts)
13:  ▷ Weighted costs allow for favoring smaller packet size/penalize larger size.
14:  weightedSizeCosts ← computeWeightedSizeCosts(sizeCosts, pktSizeWeights)
15:  ▷ Based on weighted cost, decide probability of selection for each packet size.
16:  pktSizesPrb ← computeProbabilities(weightedSizeCosts, maxSizePkt, strictness)
17:  ▷ Choose the output packet size using weighted selection probabilities.
18:  chosenPktSize ← choosePktSize(pktSizesPrb, maxSizePkt)
19:  ▷ Update current distribution.
20:  currentDistCounts[lastNPkts][chosenPktSize] + +
21:  currentDistCounts[lastNPkts][totalPkts] + +
22:  lastNPkts.enqueue(chosenPktSize)
23:  doWhisper(currentPkt, chosenPktSize)
The cost for each potential packet size represents the distance between the target and current distribution, if that particular packet size was chosen. Line 13 modifies the cost of choosing a packet size for the next packet based on a \textit{pktSizeWeights} weighting parameter for each target packet size. The weighted costs allow for favoring smaller packet sizes while penalizing larger packet sizes. This allows us to trade off between performance and security. Based on weighted cost, line 15 assigns the probability of selection to each packet size. The non-negative \textit{strictness} parameter determines how strictly the target distribution adheres to the superdistribution. A smaller value means stricter adherence compared to a larger value. The strictness parameter allows the mapping algorithm to boost the selection probability of smaller packet sizes to reduce the bandwidth overhead by trading-off security. Line 17 returns either the next output packet size based on the computed probabilities or the size of the maximum packet if there are no non-zero probabilities. Lines 19-22 update the current distribution counts and the last \( n \) packets. Line 23 then modifies the input audio frame (by overlaying tuning audio) before passing it to the VoIP application such that the size of the encoded output packet generated by the VoIP application matches the desired packet size chosen in line 17.

5.4.3 \textbf{Whisper}

Whisper's overarching goal is to decrease the accuracy of re-identification attacks by modifying the size of encoded packets generated by the VBR codec of a VoIP application. The modification of audio frames to produce packet sizes that are reflective of the superdistribution minimizes information leakage and reduces the accuracy of traffic re-identification attacks.
As shown in Figure 5.2, Whisper mitigates re-identification attacks by over-laying extra audio, called *tuning audio*, to the audio frames generated by the sender before they are passed to the VoIP application.

In our preliminary investigation, we observed that the addition of tuning audio to the original audio can alter the size of the encoded output generated by the VBR codec. VBR codecs are sensitive to the complexity of the audio being encoded; the output data of VBR per unit time varies with the audio complexity. Encoding an audio frame containing a high frequency (ultrasonic) signal will therefore result in a larger encoded packet size as compared to an audio frame with silence. We leverage this behavior of VBR codecs to overlay tuning audio to alter input audio frames in order to achieve the desired size of the encoded output, as determined using the superdistribution (see §5.4.1 and §5.4.2). As we discuss in the remainder of this section, we consider various forms of tuning audio.

### 5.4.3.1 Characteristics of tuning audio

Whisper affects packet sizes by adding tuning audio to the sender’s audio messages before they are encoded by the VoIP application. On the receiver side, the VoIP application decodes the encoded audio, which includes the original audio frames intermixed with the tuning audio. Whisper is a unilateral defense and does not require any support on the receiver; put equivalently, the receiver does not attempt to actively remove the tuning audio. This restricts the types of tuning audio that can be used, since audible tuning audio could significantly degrade audio quality. In contrast, tuning audio should not introduce extraneous noise and have minimal impact on the receiver’s perceived audio quality.

For example, even if using white noise as tuning audio results in the desired output packet size for a given audio frame, the white noise will be audible in the
decoded audio on the receiver side and will too substantially degrade the quality of
the communication.

To satisfy these requirements, we consider tuning audio that lies on and beyond
the boundary of the human auditory range (20 Hz to 20 kHz [90]). Even though
frequencies outside this range are imperceptible to human listeners, we found that
they are not discarded by popular VBR encoders. Moreover, their inclusion as tuning
audio influences the size of the encoded output, without introducing any perceptible
noise in the decoded output on the receiver side. We posit that the codecs typically
err on the side of preserving audio quality and are inexact in their filtering. We leave
it at future work to explore such behavior of VBR codecs.

In addition to inaudible frequencies, we also consider extremely low amplitude
tuning audio signals in the audible frequency range. This was necessitated by the
observed relationship between the range of input frequencies in the input audio
to be encoded and the corresponding encoded packet size. We observed that when
using the Opus codec [43], for instance, there were some transitions from one packet
size to another, as required by the superdistribution, that we could not achieve by
injecting inaudible tuning audio. These required transitions from input packet sizes
to target packet sizes were such that the use of tuning audio below 20 Hz resulted
in encoded packet sizes less than the desired packet size, whereas the use of tuning
audio above 20 kHz resulted in encoded packet sizes greater than the desired encoded
packet size. Thus, to achieve these target packet sizes, we found it necessary to inject
low amplitude (volume) tuning audio. We further discuss the use of various types
of tuning audio and their implications to security, audio quality, and bandwidth
overhead in §5.5.
**Figure 5.3**: Whisper’s workflow of modifying an input audio frame using tuning audio.

### 5.4.3.2 Whisper Workflow

Figure 5.3 shows Whisper’s high level workflow. To protect a speaker’s VoIP communication from traffic re-identification attacks, the overlaying of tuning audio onto the outgoing audio frame should happen before it is encoded by the codec. Our user model assumes that Whisper has access to an implementation of the codec used by the VoIP application. Using the standalone codec implementation, Whisper first encodes the input audio frame $a_q$ generated by the sender to determine the encoded packet size $X \in \Sigma$. It then uses the MorphStream procedure (Algorithm 1) to determine the target packet size $T \in \Sigma$ (where $T \geq X$) for the audio frame. If the encoded packet size $X$ matches the target packet size $T$ required by MorphStream (i.e., $X = T$), then no change is required to the size of the encoded packet and there is no need for any tuning audio overlay. In this case, Whisper trivially outputs the unmodified input audio frame $a_q$ as the output.
In the case in which the encoded packet size $X$ of the input audio frame $a_q$ does not match the desired packet size $T$, Whisper overlays a single tuning audio from a predetermined candidate set (explained below) onto $a_q$, encodes the modified frame $a'_q$ using the standalone codec implementation and determines the encoded packet size $X'$ of $a'_q$. If the encoded packet size $X'$ equals the desired packet size $T$, Whisper outputs the modified audio. Otherwise (i.e., $X' \neq T$), Whisper tries the next tuning audio.

We restrict the time Whisper can take to try different tuning audio from the set of candidates to prevent gaps in audio on the receiver side. In our evaluation, the codec encodes 20 ms of audio per outgoing packet. Therefore, we limit Whisper to try different tuning audio within 20 ms. This restricts the number of tuning audio candidates that can be tried as overlays to achieve the desired packet size $T$.

If the encoded target packet size $T$ is not achieved within 20 ms, Whisper outputs an audio frame according to a fallback strategy: in the default strategy, Whisper outputs the unmodified audio frame $a_q$; the random strategy overlays the input audio frame $a_q$ with tuning audio selected uniformly at random from the candidate set (see below); finally, the max strategy outputs the input audio frame overlaid with a high frequency tuning audio such that the resulting encoded packet is maximally sized (i.e., $\Sigma_{|\Sigma|}$). We analyze the impact of the various fallback strategies in §5.6.1.

### 5.4.3.3 Generating the tuning audio candidate set

To build the pool of tuning audio candidates, we use the Sox utility [46] to produce audio tones that are 20 ms in duration and are composed of one or more sine wave signals at different frequencies and amplitudes. We first consider candidates that lie outside or at the boundary of the human auditory range. In particular, we consider the infrasonic integer valued frequencies between 1 and 18 Hz, and the
four ultrasonic frequencies between 20-23 kHz, at increments of 1 kHz. For each frequency, we generate multiple tuning audio candidates with different peak amplitudes, spaced uniformly, with a maximum peak amplitude factor of 0.5 (meaning, one-half the original amplitude of the sine wave).

As discussed above, the use of tuning audio in the inaudible range fails to achieve certain transitions between source and target packet sizes. Thus, we also include candidates with frequencies within the human audible range, but with peak amplitudes factors not exceeding 0.001. This ensures that the tuning audio that lie within the human audible range remain faint in comparison to the actual audio produced by the human speaker. Within the audible range, we consider 40 equally spaced frequencies between 100 Hz and 20 kHz as candidate tuning audio.

Finally, we also consider tuning audio candidates that are composed of sine waves at three to five randomly chosen frequencies.

This results in a (rather large) set of tuning audio candidates. This is undesirable since Whisper needs to identify the correct tuning audio to overlay to achieve the desired packet size (via trial-and-error) within 20 ms. To prune the set of tuning audio candidates, we select a random subset of the training corpus and construct a superdistribution over all audio samples in this subset. We then morph this subset using all the tuning audio candidates in the pool of candidates. For each audio frame to be encoded, we consider every candidate tuning audio in the existing pool to encode each frame until we hit the desired packet size for that frame or run out of tuning audio to try. The candidates are tried in the order of ultrasonics, infrasonics and then those within the human audible range. For each of these categories, we try tuning audio in the increasing order of peak amplitude to prioritize quieter candidates. Starting with a large set of tuning audio candidates and encoding the subset of training data, we note the number of successful transitions achieved by the current
pool. We also note the number of successful transitions achieved by each candidate tuning audio. To shrink this pool of tuning audio such that all candidates in the pool can be overlaid and encoded within 20 ms, we repeatedly eliminate the tuning audio with the minimum number of successful transitions each from the ultrasonic, infrasonic and audible range candidates. During our shrinking process, we found that the candidates with the same frequency (outside of the human audible range) but with peak amplitude difference of less than 0.3 resulted in the same encoded packet size for a given input packet. This allowed us to further prune the pool by eliminating tuning audio with nearby peak amplitudes for a given frequency without affecting the total number of successful transitions.

The above procedure produces a final set of 64 inaudible candidates that lie outside or at the boundary of human and 87 audible candidates. All of the tuning audio are faded-in and faded-out to prevent the appearance of “clicking” noise across frame boundaries in the decoded output. This smoothing is necessary at frame boundaries to compensate for physical limitations in commodity speakers: speakers feature diaphragms with specific frequency response ranges that cause artifacts (clicks) when inter-frame transitions are insufficiently smooth.

5.5 Evaluation

We next evaluate the efficacy of Whisper to defeat re-identification attacks and examine the defense’s communication overheads and effects on audio quality.

5.5.1 Experimental setup

We use a subset of the Voxforge speech corpus [51] for evaluating Whisper. Our dataset is comprised of 21 speakers (14 male and 7 female) reading English liter-
ature recorded under different settings and with various background noises. The heterogeneity in recording environments influences the VBR codec’s encoding behavior, making this a conservative (difficult) case for traffic morphing defenses such as Whisper. For each speaker, we consider 240 audio samples.

We use the Opus codec [43] to evaluate our proposed defense. The Opus codec is standardized by the IETF and is the successor of the Silk codec considered in prior work [117]. We encode our training corpus with Opus in VBR mode with its default parameters to generate encoded packets of various sizes. We note that the number of distinct packet sizes ($|\Sigma|$) generated by the Opus codec is far more than the Speex and Silk codecs considered in previous research [117, 161]. Speex and Silk produced only nine and eight distinct packet sizes respectively, whereas the Opus codec outputs a much larger range of packet sizes which is dependent on the sampling rate of the input audio. All audio samples in our dataset were sampled at 16 kHz, resulting in encoded packets with a contiguous packet size distribution between 62 to 327 bytes.

5.5.2 Evaluation Strategy

We evaluate the effectiveness of our proposed defense by comparing the attacker’s ability to successfully perform a traffic re-identification attack on Opus-encoded VoIP streams when (i) no defense is applied and (ii) Whisper is enabled. The attacker’s goal is to successfully identify the speaker (out of the 21 speakers in our dataset) from the intercepted packet stream. Figure 5.1 shows the high level overview of traffic re-identification attacks on VoIP streams.

Since the large number of distinct packet sizes generated by Opus makes traffic analysis difficult, we adopt a binning strategy to reduce the number of distinct packet sizes, mapping the various packet sizes into eight bins prior to performing traffic analysis.
analysis. (That is, we force $|\Sigma| = 8$.) We consider the relative frequency of $n$-grams as features for the machine learning classifier. We considered various supervised machine learning classifiers and $n$-gram features during our investigation and found trigram features with an SVM classifier to provide the best accuracy. We, therefore, report results for 10-fold cross validation with an SVM classifier that uses the relative frequency of various trigrams as the feature vector.

We provide the attacker with access to the same training corpus used by Whisper to generate the superdistribution. This conservative assumption only provides more power to the attacker for improving its classifier. The attacker is also allowed to train or update its existing classifier with packet streams generated by Whisper. That is, the attacker is Whisper-aware and can apply Whisper as a preprocessor over the training corpus, allowing it to train on (labeled) Whisper-processed traffic streams. As discussed in §5.4.3, we make use of the *inaudible* and the *audible* sets of tuning audio in our evaluation.

### 5.5.3 Attack Accuracy

We define the *attack accuracy* to be the average accuracy across the ten folds of the cross validation. The *best case attack accuracy* (from the attacker’s perspective) corresponds to the maximum accuracy achieved by the attacker using its SVM classifier, across all tested configurations (e.g., superdistribution parameters and fallback schemes).

#### 5.5.3.1 Baseline accuracy

When no defense is applied, the attacker can perform traffic analysis of Opus-encoded VoIP packet streams and identify the speaker from the dataset with a best
case attack accuracy of 97.22%, using trigrams and an SVM classifier. This shows that the Opus codec in VBR mode is vulnerable to traffic re-identification attacks.

### 5.5.3.2 Whisper accuracy

Whisper is able to significantly reduce the attacker’s accuracy of traffic re-identification. Using candidates from the inaudible set of tuning audio as overlays, the attacker’s best case attack accuracy is reduced to 62.24% (compared to the baseline case of 97.22%). With the use of tuning audio from both the inaudible and audible set, the best case attack accuracy is further reduced to just 31.13%. The union of inaudible and audible sets outperform the smaller inaudible set because the tuning audio in the inaudible set are only able to morph packets to certain packet sizes, whereas the inaudible and audible tuning audio together are able to cover the entire range of target packet sizes.

We also compare Whisper’s effectiveness to a hypothetical technique that is able to perfectly morph the distribution of packet sizes in the input audio stream to that of the superdistribution. (The approach by Moore et al. [117] is always successful at morphing to the superdistribution, but does so at the expense of requiring bilateral cooperation between the two communicating parties.) Whisper fails to achieve ideal morphing when it cannot find a tuning audio from the candidate set of tuning audio that results in the target packet size within 20ms; in such cases, it uses one of the fallback schemes to modify the audio frame (see §5.4.3). Notably, however, such failures are rare and have only a modest effect on the defense’s effectiveness: perfect morphing to superdistribution achieves the best case attack accuracy of 26.3%, compared to 31.13% when Whisper is used. We discuss the effects of various fallback strategies in more detail in §5.6.1.
5.5.4 **Bandwidth Overhead**

The bandwidth overhead incurred by Whisper stems from the increase in packet sizes necessary for hiding the underlying packet size distribution. Whisper incurs modest overheads of 34.01% and 38.43% (relative to unprotected audio) with inaudible tuning audio and the union of inaudible and audible sets of tuning audio, respectively. As a point of comparison, switching to constant bitrate encoding imposes nearly a 90% overhead. Whisper allows for tunable security and performance, with one coming at the cost of decreasing the other. For example, the minimum bandwidth overhead using both the inaudible and audible tuning audio candidate sets can be reduced from 38.43% to 18.6%, at the cost of increasing the accuracy of re-identification attacks from 31.13% to 52.94%. We discuss these tradeoffs in more detail in §5.6.2.

5.5.5 **Impact on Audio Quality**

We evaluate the impact of adding the tuning audio on audio quality, as measured on the receiver side. We use the following two methods to quantify VoIP quality:

5.5.5.1 **Virtual Speech Quality Objective Listener (ViSQOL)**

ViSQOL [92] is a model of human sensitivity to degradations in speech quality. It uses a spectro-temporal measure of similarity between a reference and a test signal to determine the quality of speech in an audio sample and provides a mapping from an internal metric to a Mean Opinion Score (MOS) estimate. The MOS metric [40] has been commonly used to measure the quality of audio, including VoIP conversations. The metric ranges from a quality score of 1.0 to 5.0, with 1.0 being the worst. Actual VoIP calls usually lie in the range of 3.5 to 4.2 [39]. To determine the impact of
tuning audio on audio quality, we use the reference implementation made publicly available by Hines et al. [92]. We refer to the MOS estimate generated by this implementation as the ViSQOL score.

We consider each audio sample from our training corpus. As a baseline, for each audio sample, we compare the raw audio without any VBR encoding to itself. Unsurprisingly, this yields an average quality score of 5.0 across the entire dataset, as the reference and test audio samples are identical.

We next assess the quality achieved after encoding with the Opus VBR codec. Equivalently, this is the audio quality that results when the Whisper defense is not used. Here, we compare each raw audio sample to the sample produced after encoding with Opus. This yields an average ViSQOL audio quality score of 4.6. We consider this a reasonable “upper-bound” for defenses against re-identification attacks.
Figure 5.4 shows the cumulative distribution of ViSQOL scores across all audio samples in the corpus for no encoding (“No Codec”), Opus without any Whisper protections (“Opus”), and Whisper. For the Whisper configuration, we use both inaudible and audible tuning audio, since they offer the best security (corresponding to a best case attack accuracy of 31.13%) but also intuitively should impose the greatest degradation in audio quality (since it inserts audible noise).

When Whisper uses both inaudible and audible tuning audio, we observe an average ViSQOL score of 3.9; the average increases slightly to 4.0 when Whisper uses only the inaudible tuning audio. In summary, Whisper imposes a modest degradation in audio quality, and the difference between using inaudible tuning audio with or without the audible tuning audio is minimal.

The minor difference in ViSQOL scores between the inaudible and the union of inaudible and audible tuning audio settings indicates a potential downside in using automated models to measure audio quality: such techniques do not satisfactorily filter out audio outside of the human audible range, and thus may not reflect how actual human listeners perceive audio quality. That is, they may be too conservative because they do not fully model human hearing limitations. This motivates our subjective, human-based assessment of audio quality, which we describe next.

5.5.5.2 User Study

To further understand the impact of Whisper on the quality of decoded audio, we conduct a small user study that asks human evaluators to rate the quality of a given audio sample. For the user study, we randomly choose eight audio samples—with four female and four males speakers—from our dataset. For each of these eight samples, we also select the corresponding audio files produced with Opus without Whisper and with Whisper. For the Whisper-encoded version, we select
the candidates encoded with the \( \text{max} \) fallback option, with the packet cost weight ratio between adjacent packet sizes set to 1 and the strictness parameter set to 0 (see §5.4.2). Thus, we use a total of 24 audio files in our user study encoded in three ways. The audio samples presented to the human evaluators are available at https://www.whisperIntoVoIP.com.

Figure 5.5 illustrates the design of our online user survey. In Part A of the survey, the participants first listen to an audio sample and are asked to transcribe it. This ensures that the participants actually listened to the audio and also informs us whether they are able to understand the spoken audio content. The participants are then asked to rate the overall audio quality on a five point Likert scale from Bad to Excellent (or Excellent to Bad, to minimize ordering effects). They are then asked to briefly explain their choice of rating as a free text response.

Part B of the survey asks the same questions as Part A but for an audio that differs in the encoding method and the spoken content from the audio presented in Part A. Finally, Part C concludes the survey with demographic questions about education, gender, ethnicity, age, income, and employment.
Table 5.1: Participant demographics for the user study to determine audio quality.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gender</td>
<td></td>
</tr>
<tr>
<td>Female</td>
<td>32.9%</td>
</tr>
<tr>
<td>Male</td>
<td>67.1%</td>
</tr>
<tr>
<td>Ethnicity</td>
<td></td>
</tr>
<tr>
<td>Caucasian</td>
<td>75.9%</td>
</tr>
<tr>
<td>African American</td>
<td>6.5%</td>
</tr>
<tr>
<td>Hispanic or Latino</td>
<td>11.6%</td>
</tr>
<tr>
<td>Asian</td>
<td>5.8%</td>
</tr>
<tr>
<td>Other</td>
<td>4.4%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td></td>
</tr>
<tr>
<td>18-29 years</td>
<td>39.4%</td>
</tr>
<tr>
<td>30-49 years</td>
<td>53.3%</td>
</tr>
<tr>
<td>50-64 years</td>
<td>6.5%</td>
</tr>
<tr>
<td>65+ years</td>
<td>0.7%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Education</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>H.S. or below</td>
<td>13.1%</td>
</tr>
<tr>
<td>Some college</td>
<td>24.8%</td>
</tr>
<tr>
<td>B.S. or above</td>
<td>62.0%</td>
</tr>
</tbody>
</table>

Percentages may not add to 100% due to non-response or selection of multiple options.

Table 5.2: Number of responses for each type of audio depending on how the audio was encoded.

<table>
<thead>
<tr>
<th></th>
<th>No Codec</th>
<th>Opus</th>
<th>Opus with Whisper</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responses</td>
<td>93</td>
<td>89</td>
<td>92</td>
</tr>
</tbody>
</table>

Recruitment. We used Amazon’s Mechanical Turk (MTurk) crowdsourcing service to recruit participants for the user study. We required participants to be at least 18 years old, fluent in English, and located in the United States. To improve data quality, we also required participants to have at least a 95% HIT approval rate [128]. Participants were paid $1.00 for completing the study, which was reviewed and approved by our institution’s ethics board. The demographics of our participants are summarized in Table 5.1.

Results. In total, 150 MTurk human evaluators participated and completed our study. We exclude three responses as duplicates based on their originating IP addresses and only consider their first response. We also discard 10 responses that
Figure 5.6: Audio quality as reported by human evaluators for audio with no encoding and audio encoded with the Opus codec with and without Whisper defense.

provided unintelligible transcriptions. For the remainder of this chapter, we refer to the remaining 137 survey participants. Table 5.2 shows the number of responses for each type of audio presented.

Figure 5.6 summarizes the audio quality as reported by the survey participants. For the baseline audio with no encoding, the participants reported 4.2 as the average audio quality. When the audio was encoded with Opus (without Whisper protections), the average audio quality was 4.4. The participants, therefore, did not perceive any significant difference in the quality of audio when encoded with the Opus codec.

For audio encoded with Opus and protected using Whisper, participants reported an average audio quality of 3.6. On examining the reasoning behind the responses, one participant reported that he could hear a bit of static in the background but everything else was clear. Another participant said that there was noise in the background but could fully understand the audio. We remark that all of the study participants were able to correctly understand the contents of the spoken audio, even when they reported hearing artifacts or background noise. The perceived
audio quality reported by the participants of the user study indicates that Whisper has no effect on listeners’ ability to understand the audio and only introduces minimal noise.

**Comparison of ViSQL and User Study Results.** The results obtained from the ViSQL metric and the user study are largely consistent. For example, for audio encoded with the unprotected Opus codec, the audio quality reported by the ViSQL metric (4.6) is close to that obtained from the user study (4.4). Similarly, both approaches report an average audio quality of 3.6 for audio encoded with Opus equipped with Whisper. Overall, our two techniques are consistent in showing that Whisper does not significantly impact audio quality and does not affect the perception of audio content.

## 5.6 Tuning Whisper

Whisper has a number of configuration parameters that influence its effectiveness in thwarting re-identification attacks, its impact on audio quality, and its bandwidth overheads. In this section, we highlight some important points in this parameter space.

### 5.6.1 Effect of Fallback Options

Whisper overlays tuning audio on input audio before it is encoded by the sender’s VBR codec; the choice of tuning audio is dictated by the target packet size as determined using the superdistribution. As discussed in §5.4.3, Whisper may fail to achieve the target packet size within the 20 ms window in which it needs to modify
the audio frame. In such (rare) cases, Whisper can choose from the default, random or max fallback options (see §5.4.3 for details).

Figure 5.7 shows the effect of the various fallback options on attack accuracy, bandwidth overhead, and audio quality (note that lower is better for the first two performance metrics). Overall, the choice of fallback option has only a minor effect on the three performance metrics. Falling back to the maximum (max) packet size only slightly reduces the attack accuracy while incurring slight bandwidth overhead. Audio quality, as measured using ViSQOL, also remains almost the same across different fallback options. Thus, a user can safely configure Whisper to use any of the fallback options.

5.6.2 Effect of MorphStream Parameters

The MorphStream procedure (Algorithm 1) determines the output packet size for each input audio frame based on the superdistribution. The user specifies the pkt-SizeWeights and the strictness parameters to favor security or performance.
Figure 5.8: Effect of different \texttt{pktSizeWeights} values on various performance metrics.

Figure 5.8 shows the effect of the cost weight ratio between adjacent packet sizes on various performance metrics when the default fallback scheme is used. As discussed in §5.4.2, the \texttt{pktSizeWeights} parameter influences the relative cost of choosing a packet size among all target packet sizes. By increasing the relative cost between packet sizes, the cost of selecting a larger packet size increases, resulting in a comparatively lower bandwidth overhead as smaller-sized output packets become more favorable. This also results in increased attack accuracy for the attacker as MorphStream may now choose a smaller packet size which can result in a packet size distribution that does not closely resemble the superdistribution. However, as shown in Figure 5.8, these effects are small. When \texttt{pktSizeWeights} is set to the maximum tested value, the attack accuracy rose to approximately 40% while providing little bandwidth savings. This indicates that a reasonable value of \texttt{pktSizeWeights} is 1, maximizing the efficacy of the attack while imposing little bandwidth overheads.
Figure 5.9: Effect of strictness parameter on performance metrics.

Figure 5.9 shows the effect of the strictness parameter on various performance metrics. The non-negative strictness parameter determines how strictly the target distribution should match the superdistribution. A smaller value results in stricter adherence to the superdistribution, achieving greater security. As the strictness parameter increases, the MorphStream procedure boosts the selection probability of smaller packet sizes, even though it may cause the target distribution to stray from the superdistribution. The strictness parameter allows the user to trade off between security and bandwidth savings, but (as shown in the Figure) does not affect the decoded audio quality. We consider a default value of 0 for the strictness parameter as it does not allow deviation from the superdistribution thus providing maximum security while incurring modest bandwidth overhead.
5.7 **Summary**

In this chapter, we propose the first unilateral defense, Whisper, for thwarting traffic analysis of encrypted VoIP streams. One of the major limitations of previously proposed blackbox defenses is that they require support from both the sender and receiver sides of a VoIP stream; that is, both of the communicating parties’ VoIP clients must support the defense. Unfortunately, to our knowledge, no such VoIP client has implemented existing defenses. In contrast, Whisper enables unilateral protections that can be deployed by either communicating party, without requiring the participation of the other and without modifying the VoIP client. Whisper is thus compatible with existing closed-source VoIP software.

Building on existing work, and leveraging the mechanisms of audio perception in humans, Whisper uses tuning audio at the boundaries of the human audible range to manipulate the size of the audio frames generated by VBR codecs. Our experiments demonstrate that Whisper significantly degrades the accuracy of re-identification attacks while incurring only a small loss in audio quality. Additionally, Whisper preserves much of the bandwidth savings of VBR.
Chapter 6

Limiting Privacy Exposure via Voice Input

Voice synthesis uses a voice model to synthesize arbitrary phrases. Advances in voice synthesis have made it possible to create an accurate voice model of a targeted individual, which can then in turn be used to generate spoofed audio in his or her voice. Generating an accurate voice model of target’s voice requires the availability of a corpus of the target’s speech.

This chapter makes the observation that the increasing popularity of voice interfaces that use cloud-backed speech recognition (e.g., Siri, Google Assistant, Amazon Alexa) increases the public’s vulnerability to voice synthesis attacks. That is, our growing dependence on voice interfaces fosters the collection of our voices. By leveraging the process of audio understanding in machines, we show that voice recognition and voice accumulation (that is, the accumulation of users’ voices) are separable. This chapter introduces techniques for locally sanitizing voice inputs before they are transmitted to the cloud for processing. In essence, such methods em-
ploy audio processing techniques to remove distinctive voice characteristics, leaving only the information that is necessary for the cloud-based services to perform speech recognition. Our experiments show that our defenses prevent state-of-the-art voice synthesis techniques from constructing convincing forgeries of a user’s speech, while still permitting accurate voice recognition.

6.1 Motivation

A person’s voice is an integral part of his or her identity. It often serves as an implicit authentication mechanism to identify a remote but familiar person in a non-face-to-face setting such as a phone call. The ability to identify a known person based on their voice alone is an evolutionary skill (e.g., enabling a child to quickly locate its parents) and is an ingrained and automated process that requires little conscious effort [113]. That humans regularly authenticate each other based solely on voice lends to a number of potential impersonation attacks, which notably include voice spear phishing and various other forms of social engineering. The ease at which such attacks can be conducted has increased due to advances in speech synthesis. Emerging services such as Adobe Voco [6], Lyrebird.ai [19, 38] and Google WaveNet [10] aim to produce artificial speech in a person’s voice that is indistinguishable from that person’s real voice. Surprisingly, producing believable synthetic speech does not require a large corpus of audio data. For example, it has been reported that Adobe Voco can mimic a person’s speech with as little as 20 minutes of the targeted speaker’s recordings [6, 7], and Lyrebird.ai can create a digital version of a voice from a one minute speech sample.

Advances in voice synthesis open up a large number of potential attacks. An adversary who has access to a speech sample of a target victim could apply voice syn-
thesis to authenticate as the victim to banks and other commercial entities that rely on voice authentication [5, 12, 23]. Forged speech could also be used to impugn reputations (e.g., for political gain) or plant false evidence. In general, voice synthesis poses a significant security threat wherever voice is used as an authenticator.

A core requirement of such attacks is that the adversary must have access to a corpus of voice recordings of its target.

The ability to obtain such samples is buoyed by the rising popularity of voice input. Voice input has become ubiquitous and a common method of computer-human interaction, in no small part because it is a natural (to humans) method of communication. Smartphones, tablets, wearables and other IoT devices often come equipped with voice assistants (VAs) such as Alexa, Siri, Google Now and Cortana. Dedicated VA devices such as Amazon Echo and Google Home have found their way into living rooms, constantly listening to users’ voice input and providing quick responses. Users of these devices regularly surrender their voice data, making them more vulnerable to future voice synthesis attacks.

Currently, only the voice assistant service providers have access to the voice samples of a user. However, it is unclear due to conflicting reports whether the application developers will get access to user’s voice samples [13, 15]. For example, it had been reported that Google Home allowed access to raw voice command audio to application developers while Amazon Echo also plans to do so in the future [15]. Thus, the increased use of voice input increases the opportunities to gain access to raw voice samples of the users.

The chapter proposes defenses to reduce the threat of voice synthesis attacks for ordinary users. We concede that much voice data is already in the public domain—certainly, it is not difficult to obtain audio recordings of celebrities and politicians, or of ordinarily users who post their own video or audio content to publicly accessible
social media (e.g., YouTube). Such users are already vulnerable to voice synthesis attacks and the techniques that we propose in this chapter unfortunately do not attempt to protect them. Rather, our aim is to present wide-scale vulnerability to voice synthesis attacks by changing the norm – that is, by permitting the use of voice-based services (e.g., VAs) while preventing the collection of users’ raw (unmodified) voice inputs.

We propose a defense that prevents an adversary with access to recordings of voice commands, issued by users to VAs, from building a voice model of a targeted user’s voice. Our approach is based on the following two observations:

1. *A user does not need to sound like herself to use a voice assistant.* The first step in generating a response to a user’s voice command is conversion of speech to text, i.e., speech recognition. Modern speech recognition systems are oblivious to unique characteristics of a person’s voice, and thus, are able to transcribe audio from thousands of users. Therefore, altering a user’s voice so that it does not sound like the user herself does not prevent her from using VAs.

2. *Speech recognition systems do not need all the information present in spoken audio.* The first step in speech recognition is usually a feature extraction step that converts the high dimensional input audio into low dimensional feature vectors which are then used as inputs to machine learning models for transcribing the audio. Removing some of the information from the high dimensional audio, that is anyway thrown away during the feature extraction, will not affect the speech recognition process but can be used to alter the voice characteristics of the audio.

---

1 The speaker based personalization supported by various VAs is not hampered by such alteration, since the speaker detection is done locally on the client device (e.g., smartphone) and only applies to the activation keywords for the voice assistants.
In brief, our proposed defense extracts audio information from voice commands that are relevant for speech recognition while perturbing other features that represent unique characteristics of a user’s voice. Put plainly, we strip out identifying information in audio, which significantly hinders (if not makes impossible) the task of speech synthesis. Our approach could be applied locally—in particular, on smartphones and smartspeaker devices—as a “security filter” that prevents third parties (whether they be the speech recognition service itself, third-party developers, or even network eavesdroppers) from being able to construct convincing synthesized voices. Additionally, our proposed defense has the benefit that it does not require any modifications to the cloud-based speech recognition systems.

In what follows, we describe our initial design and prototype of our defense. Our preliminary experiments, including a small (IRB-approved) user-study, show that our proposed approach prevents the constructing of convincing voice synthesis models while imposing minimal effects on the accuracy of speech recognition.

6.2 Background

We believe we are the first to propose filtering raw voice audio data for the purposes of thwarting voice synthesis attacks. However, existing work has proposed several approaches for achieving privacy-preserving voice recognition:

Smaragdis et al. [143] propose a privacy-preserving speech recognition system as an instance of secure multiparty computation, where one party (the transcriber) has a private model for performing speech recognition while the other parties have private audio data that need to be transcribed without revealing the audio content to the transcriber. However, their work does not describe the performance or accuracy.
of such a system and is limited to HMM-based speech recognition systems. Additionally, secure-multiparty computation is computationally expensive and requires both parties to cooperate. In contrast, our approach can be deployed locally and does not require any changes to existing speech recognition services.

Pathak et al. [125] provide a number of techniques for privacy-preserving speech processing. They describe various frameworks that aim to make conventional speech processing algorithms based on statistical methods, such as HMM, privacy-preserving by computing various operations via secure operations such as secure multiparty computations, additive secret sharing, and secure logsum. Their techniques are impressive, but suffer from practical limitations due to their dependence on computationally expensive cryptography. Their framework also does not achieve good speech recognition accuracy; in contrast, our defense is intended for advanced and (arguably) accurate services such as Google’s, Apple’s, and Microsoft’s cloud-based speech recognition systems.

Ballesteros and Moreno propose scrambling of a private speech message to a non-secret target speech signal using a secret key, which the receiver unscrambles using the same shared secret [57]. The target speech signal’s plaintext is different from that of the secret message, so as to fool an eavesdropping adversary. However, the technique requires both cooperation between the sender and receiver of the scrambled signal as well as out-of-band key sharing.

More generally, techniques that prevent speech recognition services from learning the transcription (e.g., via secure multiparty computation) are not applicable to our problem, since in our setting, transcriptions are required by the service provider to respond to voice commands. All major existing VA systems (including Google Home, Amazon Alexa, and Siri) use proprietary, cloud-based speech recognition; it
is unlikely that these services would choose to deploy expensive and poorly scalable cryptographic-based protocols.

In contrast, our proposed defense aims to improve the privacy of voice data for existing and already deployed systems that are widely used by millions of users worldwide without requiring any changes to the speech recognition systems.

Various voice changing applications [50] and vocoders such as Cylon Centurion [31] morph spoken audio into various sounds such as a robot’s voice, etc. However, these applications are not designed to provide privacy and the modifications made to the audio are usually reversible [50]. Furthermore, speech recognition systems perform poorly on modified audio generated by such vocoders. This is because speech recognition systems are only trained to understand human sounding audio. In contrast, our proposed defense modifies the audio while preserving the acoustic information required by speech recognition systems.

Most relevant to the work presented in this chapter are the studies discussed in Chapter 3 done by the author and other co-authors [65, 152], that show the feasibility of specially crafting audio that is intelligible to computer speech recognition services but not to human listeners. The defense proposed in this chapter borrows the use of MFCCs to extract audio information from the earlier approach, and removes additional audio features that provide uniqueness to a person’s voice.

### 6.3 Threat Model

We consider an adversary whose goal is to impersonate a targeted individual’s voice. The adversary achieves its goal of generating spoofed audio in the target’s voice by building an accurate model of his voice by using speech synthesis services such as
Adobe Voco or Lyrebird.ai. Crucially, to be successful, the adversary needs to first collect a corpus of the target user’s speech.

6.3.1 Acquiring voice samples

Our threat model assumes that the adversary requires high quality voice speech samples of the target to build its voice model. As an example means of collecting voice samples, an adversary could create a (legitimate) voice application\(^2\) for a voice assistant, which provides raw voice command audio data to the application. Alternatively, a speech recognition service may itself be malicious and/or sell users’ speech data to other parties. Finally, if speech is transmitted unencrypted (which hopefully is a rarity) during a voice-over-IP call, a network eavesdropper could trivially collect a corpus.

We emphasize that in this work, we explicitly do not consider voice collection from in-person conversations, postings of audio on social media websites (e.g., YouTube), broadcast media (e.g., TV), or other sources. We acknowledge that highly skilled and committed adversaries can likely obtain audio of a specific person, for example, by physically planting a listening device near the target. Our goal is to change the norm such that the collection of ordinary users’ audio is much more difficult. Specifically, we want to enable ordinary users to use VAs while minimizing their risk to voice synthesis attacks.

6.3.2 Generating voice models

Our threat model assumes that the adversary has access to services such as Adobe Voco or Lyrebird.ai that can be used to create a voice model of a person’s voice from the acquired voice samples.

\(^2\)These are sometimes called *skills.*
Lyrebird.ai, at its current state of deployment, is able to create a voice model of a person’s voice and synthesize arbitrary audio that share the voice characteristics of that person. We tested how well Lyrebird.ai is able to imitate a person’s speech by replaying the synthesized phrases against the speaker recognition (as opposed to speech recognition) systems that are built into personal voice assistants. Siri and Google Assistant both employ speaker recognition on their respective activation phrases “Hey Siri” and “Ok Google” to identify the active user and to provide a more personalized experience based on the user’s identity [17, 27]. The authors trained both Google Assistant and Siri on a Google Home and iPhone 8 Plus, respectively, with his voice. To ensure that the VAs were not tricked by another speaker’s voice, we successfully verified the voice assistants did not accept the respective activation phrases generated by MacOS’ say text-to-speech command. We then created a voice model of the first author’s voice using Lyrebird.ai and used the service to synthesize the activation keywords. Both of the synthesized phrases were successfully able to trick Siri and Google Assistant into believing that the phrases were spoken by the registered user. Although this is an admittedly small experiment and we acknowledge that much more sensitive voice authentication systems exist, it demonstrates the feasibility of defeating widely deployed speaker recognition systems—in particular, those that guard our smartphone devices.

6.4 **Strawman Solution: Client-side speech recognition**

We can trivially prevent an adversary from getting access to voice data by performing only *client-side* speech recognition. However, there are various practical challenges that prohibit such a solution:
Cloud-based speech recognition allows for large, complex models to be trained, deployed, and updated transparently without affecting client-facing services. Such speech recognition models require significant computing power since state-of-the-art systems rely heavily on computationally expensive deep neural networks. Cloud deployment also allows for constant improvements in speech recognition without requiring updates to client-side software or any service downtime for clients. Sending raw audio to remote servers also allows service providers to gather more data for improving the performance of their speech recognition systems. The majority of commercially deployed speech recognition systems use supervised machine learning techniques \[4, 9\] that can potentially benefit from access to more data for training or testing. In particular, Alexa, Siri, Google Assistant and Cortana all reportedly use recorded voice commands to improve their performance and accuracy of their voice-based service offerings \[2, 47, 48, 49\].

Additionally, existing open source client-side speech recognition tools (e.g., CMU Sphinx \[22\] and Mozilla’s DeepSpeech \[41\] generally have worse accuracy compared to current cloud-based speech recognition services \[61\]. Client devices such as smartphones and in-home assistants are usually too resource constrained to employ the better performing speech recognition techniques that are used by cloud-based services. Switching to open source client-side speech recognition will also require (1) changes to existing personal voice assistants and (2) support from their service providers.

### 6.5 Audio Sanitizer

Our high-level approach to reducing the threat of voice synthesis attacks is to make it more difficult to collect corpora of ordinary users’ voices. We introduce the concept
of an audio sanitizer, a software audio processor that filters and modifies the voice characteristics of the speaker from audio commands before they leave the client device. Altering such features transforms the voice in the audio commands that is available to the adversary, making it difficult to extract the original voice characteristics of the speaker and reducing the accuracy of the speaker’s voice model.

The unique characteristics of a person’s voice can be attributed to the anatomy of various organs involved in the process of generating the voice. To identify the audio features that capture the uniqueness of a person’s voice, we identify features used in speaker recognition to identify a speaker from his voice. Since the goal of speaker recognition is to tell users apart from each other based on their voice characteristics, we believe that modifying the features used for speaker recognition provides a good starting point for the audio sanitizer.

Speaker recognition system typically employ the following three types of features [163]:

1. Short-term spectral features: These features are extracted from short overlapping frames and correlate to voice timbre. Common spectral features include Mel-frequency cepstral coefficients (MFCCs) and linear predictive cepstral coefficients (LPCCs).

2. Prosodic and spectro-temporal features: These features include pitch, rhythm, tempo, pause and other segmental information and capture the speaking style and intonation.

3. High level features: These features represent speaker behavior or lexical clues and are usually extracted using a lexicon.

We focus on a subset of these features—namely MFCCs, pitch, tempo and pause—and modify them to alter the voice characteristics of the spoken audio. Our
perturbations are random, but are applied consistently for the audio of a given individual speaker. (Otherwise, if our modifications were randomly chosen per sample, then an adversary who collects a sufficient number of samples could recover the underlying voice characteristics by “subtracting away” the mean of the applied random distribution.)

In addition to modifying the identifying features of a speaker’s voice, we also remove the extraneous information present in the audio that is not required for speech recognition. Recall that the first step in speech recognition is feature extraction, which converts high dimensional, raw audio to low dimensional feature vectors. To preserve the acoustic information relevant for speech recognition, we first compute the MFCCs of the input audio and then convert the MFCCs back to audio signal by adding white noise [65]. Importantly, performing an MFCC and then inverting the MFCC back to an audio signal is a lossy operation that cannot be reversed (since information is lost). Here, our goal is to keep only the audio that is required for speech recognition while losing information that is useful to construct accurate voice models.

As we discuss in more detail in the next section, for each speaker, we choose a parameter for each feature such that the resulting sanitized audio has minimal voice characteristics of the speaker and is accurately transcribed by the speech recognition service.

6.6 Evaluation

We evaluate our proposed audio sanitizer by analyzing the degree to which it can degrade the quality of voice models to conduct speech synthesis attacks while simultaneously enabling accurate speech recognition.
Table 6.1: Modifications performed to various features by the audio sanitizer.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Modification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pitch</td>
<td>Shift up or down by 0 - $\frac{1}{5}$th octave</td>
</tr>
<tr>
<td>Tempo</td>
<td>Change by 85% - 115%</td>
</tr>
<tr>
<td>Pause</td>
<td>Introduce 0 - 15ms of pause at random 1% positions.</td>
</tr>
<tr>
<td>MFCCs</td>
<td>Nbands: 100, Numcep: 100, Wintime: 0.025s, Hoptime: 0.01s</td>
</tr>
</tbody>
</table>

6.6.1 Impact on Speech Recognition

We evaluate the impact of sanitizing audio (audio output from the audio sanitizer) by comparing the transcription accuracy of the unsanitized (unmodified) and the sanitized audio. Ideally, sanitized audio should provide identical accuracy to the baseline unsanitized audio.

We choose a random subset of 500 audio samples from the West Point Company English speech data corpus from the University of Pennsylvania’s Linguistic Data Consortium (LDC) [118]. The LDC corpus consists of both male and female, American English-language speakers, each speaking short, multiple sentences. Our subset is comprised of 130 different speakers, with 53 females and 77 males. We measure our impact on speech recognition quality using Google’s and IBM’s cloud-based speech recognition services [37, 126]. To quantify the accuracy of speech recognition systems, we consider the Levenshtein edit distance between the words of the correct, expected transcription and the best transcription provided by the speech recognition service. We report the normalized word edit distance by dividing Levenshtein edit distance by the number of words in the baseline transcription.

For each audio sample in the corpus, we first transcribe the unsanitized audio file to establish the baseline accuracy using the online speech recognition services. Each file is then sanitized using the audio sanitizer, which modifies the features that
provide unique characteristics to a speaker’s voice (see §6.5). To permanently remove the extraneous audio not required for speech recognition, we compute the MFCCs for each audio and then invert those MFCCs and add white noise to generate the sanitized audio [75]. The audio sanitizer first performs the lossy MFCC step and then modifies the pitch, tempo and pause features to produce the sanitized audio. Finally, we transcribe the sanitized audio file generated by the audio sanitizer using the online speech recognition services.

Table 6.1 shows the features and the level of modifications performed to each of those features for each audio file. For example, for male speakers, we increase the pitch by 0 to $\frac{1}{3}$th octave, randomly choosing the octave value in the specified range. To modify the tempo, we multiply the tempo of the audio by a number chosen uniformly at random from $[0.85, 1.15]$.

Figure 6.1 shows the cumulative distribution (CDF) of the normalized edit distances for the unsanitized and sanitized audio samples when using Google’s and IBM’s speech recognition services. For Google’s speech recognition service, the best-
case accuracy (i.e., having a perfect transcription and a normalized edit distance of zero) drops from 83.2% to 60.4% when the audio sanitizer is used. In the case of IBM’s speech recognition service, sanitizing the audio decreases the accuracy from 70.8% to 50.1%.

Our initial implementation of the audio sanitizer shows promise: in the worst case, transcription is perfect more than half of the time. However, we anticipate that accuracy could be significantly increased by more intelligently performing voice modifications. In particular, in our initial version of the audio sanitizer, we use a fixed set of modifications (see Table 6.1) for all speakers. Given significant variations in people’s voices, we can likely achieve improved accuracy results by analyzing individual voice characteristics and choosing specific parameter ranges on a per-speaker basis. We posit that by moving away from a one-size-fits-all model and performing per-speaker audio sanitization, we can make our sanitizer less coarse and more focused by removing only the information that makes an individual speaker’s voice distinctive.

6.6.1.1 Comparison with Client Side Speech Recognition

Figure 6.1 also shows the cumulative distribution (CDF) of the normalized edit distances for transcription accuracy achieved by DeepSpeech [41], a client side speech recognition system. The audio sanitizer is able to achieve better transcription accuracy as compared to DeepSpeech. Thus, the audio sanitizer enables users of voice assistants to reduce the privacy leakage of their voice data and without requiring any support from the service providers.
6.6.2 Privacy Gain

To conduct a speech synthesis attack, the attacker requires a corpus of the targeted user’s speech. We evaluate the efficacy of the audio sanitizer by comparing attacks’ effectiveness when the corpus is based on unmodified speech (the current norm) and speech that has been filtered by the audio sanitizer. More concretely, we examine the adversary’s ability to successfully launch an attack—that is, cause actual human listeners to conflate a synthesized voice with a legitimate recording of a speaker. To perform such an evaluation, we conduct a small user study to measure how well the attacker is able to fool human listeners when (i) using a voice model created from unsanitized voice commands and (ii) comparing that to the case in which the voice is is based on sanitized audio.

6.6.2.1 User Study

Our user study is designed to determine the success rate of an attacker when attempting to trick human evaluators with synthesized audio. The user study presents the
participants with different pairs of audio samples and asks them to specify whether they think the audio samples were spoken by the same person.

Figure 6.2 illustrates the design of our online user survey. In Part A of the survey, participants listen to two short audio samples with different speech content and are then asked about the content of the first audio as an attention check. The two audio samples are normal speech samples either from the same speaker or two different speakers, shown evenly to the participants. On the next page of the survey, participants are asked to describe the relationship between the speakers of both audio samples using a five-point scale, from “definitely spoken by same speaker (person)” to “definitely spoken by different speaker (person)”. Part A was designed to establish a baseline accuracy of how well survey participants are able to correctly identify whether two voice samples reflect the same or different speakers.

Part B of the study measures whether participants can determine the relationship between the speakers of two audio samples, when one of the audio is synthesized from a voice model. The survey participants listen to two short audio samples with different speech content. The first audio is always a normal speech audio from a single speaker, the second audio is always a synthesized audio generated from a voice model chosen based on the following two factors:

1. Voice model: the voice model is generated by either using unsanitized audio or sanitized audio.

2. Speaker: the speaker can either be same or different speaker with respect to the first audio.

Using a full factorial design, we consider the four conditions based on the above two factors for choosing the second audio in Part B as shown in Table 6.2. All voice synthesis was performed using the Lyrebird.ai service. Participants are first
Table 6.2: Number of participants assigned to each baseline condition and each of the four test conditions.

<table>
<thead>
<tr>
<th></th>
<th>Same speaker</th>
<th>Different speaker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>47</td>
<td>45</td>
</tr>
<tr>
<td>Unsanitized audio</td>
<td>23</td>
<td>25</td>
</tr>
<tr>
<td>Sanitized audio</td>
<td>22</td>
<td>20</td>
</tr>
</tbody>
</table>

asked about the content of the first audio as an attention check. On the next survey page, participants are asked to describe the relationship between the speakers of both audio samples, again on a five-point scale ranging from “definitely spoken by same speaker (person)” to “definitely spoken by different speaker (person)”.

Part B was designed to answer our primary condition of interest: i.e., while using synthesized audio constructed from a corpus of sanitized audio data, were the participants less able to identify whether the speakers were the same or different? We compare this to the case in which synthesized audio is based on normal, unmodified audio. Put simply, we determine whether the voice synthesis attacks are less convincing when they are forced to train models based only on sanitized audio samples.

In Part C, the participants again listen to the same pair of audio from Part B. They are then asked about the speech in both of the audio samples with options: “both are human voices”, “first in human voice but second is a computer generated voice”, “first is computer generated voice but second is a human voice”, “both are computer generated voices” and “not sure”. The goal of Part C was to indirectly measure how well users can identify speech that is synthesized using Lyrebird.ai.

The online survey concludes in Part D with demographic questions about education, gender, ethnicity and age.
Table 6.3: Participant demographics for the user study to determine the efficacy of audio sanitizer.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gender</td>
<td></td>
</tr>
<tr>
<td>Female</td>
<td>41.3%</td>
</tr>
<tr>
<td>Male</td>
<td>54.3%</td>
</tr>
<tr>
<td>Other</td>
<td>2.1%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ethnicity</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caucasian</td>
<td>75.0%</td>
</tr>
<tr>
<td>African American</td>
<td>8.7%</td>
</tr>
<tr>
<td>Hispanic</td>
<td>4.3%</td>
</tr>
<tr>
<td>Asian</td>
<td>7.7%</td>
</tr>
<tr>
<td>Other</td>
<td>4.3%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td></td>
</tr>
<tr>
<td>18-29 years</td>
<td>38.0%</td>
</tr>
<tr>
<td>30-49 years</td>
<td>47.8%</td>
</tr>
<tr>
<td>50-64 years</td>
<td>9.8%</td>
</tr>
<tr>
<td>65+ years</td>
<td>1.0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Education</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.S. or below</td>
<td>9.8%</td>
</tr>
<tr>
<td>Some college</td>
<td>32.6%</td>
</tr>
<tr>
<td>B.S. or above</td>
<td>55.4%</td>
</tr>
</tbody>
</table>

Percentages may not add to 100% due to non-response or selection of multiple options.

Recruitment. We used Amazon’s Mechanical Turk (MTurk) crowdsourcing service to recruit participants for the user study. We required participants to be at least 18 years old and located in the United States. To improve data quality, we also required participants to have at least 95% HIT approval rate [127]. Participants were paid $1.00 for completing the study, which was approved by the IRB at Georgetown University. The demographics of our participants are summarized in Table 6.3.

Results. In total, 104 MTurk workers participated and completed our study. Table 6.2 shows the number of responses across the baseline conditions and the four test conditions. We exclude 11 responses as duplicates based on their originating IP addresses and only consider their first response and also exclude three responses that failed the attention checks. For the remainder of this Chapter, we refer to the remaining 90 participants.

Table 6.4 summaries the results of the user study and shows the percentage of users that reported a given relationship between the speakers of the two audio
Table 6.4: Summary of responses from the user study for various conditions.

<table>
<thead>
<tr>
<th></th>
<th>Same Speaker</th>
<th>Different Speaker</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Same</td>
<td>Different</td>
</tr>
<tr>
<td>Baseline</td>
<td>76.6%</td>
<td>21.3%</td>
</tr>
<tr>
<td>Unsanitized</td>
<td>30.4%</td>
<td>60.9%</td>
</tr>
<tr>
<td>Sanitized</td>
<td>9.1%</td>
<td>81.8%</td>
</tr>
</tbody>
</table>

Each cell shows the percentage of participants that reported a given relationship (excluding the “not sure” response) between the speakers of the two audio samples for the given condition. For Baseline, the two audio were human speech, for Unsanitized and Sanitized, the first audio was human speech while the second was synthesized using Lyrebird.ai.

samples for the given condition. For the baseline response, 65.2% of the participants correctly identified the relationship between the speakers from Part A of the survey; 76.6% correctly identified the same speaker whereas 53.3% were able to correctly differentiate between two different speakers. This shows that the majority of the participants were able to correctly identify whether or not two audio samples are from the same speaker.

We next focus on the case in which the survey participants are tasked with identifying whether two samples originate from the same speaker, when one of the samples is synthetically generated using Lyrebird.ai. When the synthetic voice was produced using a corpus of the first speaker’s unmodified (non-sanitized) voice, 30.4% of the participants correctly identified that voices were from the same speaker. This corresponds to the attacker’s success rate in impersonating the targeted individual by spoofing his voice using synthesized speech generated from his voice model built using unsanitized speech audio. However, when the synthetic voice was produced using a corpus of the first speaker’s modified (sanitized) voice, only 9.1% of the participants believed that the voices were from the same speaker while 81.8% reported them to be from different speakers. Our results show that the audio sanitizer is able
to significantly reduce the efficacy of the attack; that is, the attack is far less successful when the attacker only has access to sanitized speech audio samples.

In the case of different speakers, when the synthesized voice was generated using a corpus of another speaker’s unmodified (non-sanitized) voice, 72.0% of the participants correctly identified the voices to be from different speakers while 12.0% reported them to be from the same speaker. However, the use of a sanitized audio corpus to synthesize the audio for another speaker resulted in 95.0% of the participants correctly identifying the voices to be from different speakers and none of the participants reporting the voices to be from the same speaker. In summary, the results from our user study show that given the current quality of Lyrebird.ai’s voice synthesis, an attacker with access to unmodified speech audio samples of the targeted individual can synthesize convincing spoofed speech samples in the target’s voice. However, sanitization of the audio to remove the voice characteristics prevents the attacker from generating an accurate voice model, resulting in synthesized spoofed audio that are far less convincing.

6.7 Discussion

In this section, we discuss in more detail the benefits, limitations, and deployment considerations surrounding our audio sanitizer defense.

6.7.1 Detection of Computer Generated Audio by Humans

In Part C of the online survey, we asked the survey participants to identify whether the two audio samples presented to them were spoken by a human or were computer generated. 76.7% of the participants correctly identified the first audio to be human generated speech while the second one being computer generated across all
four conditions. This shows that the users, with the current state of voice synthesis, are able to correctly identify computer generated voices. However, this does not diminish the threat posed by the collection of voice data for the purpose of building voice models for malicious purposes, since further improvements in the underlying technologies for voice synthesis and conversational voice assistants will increase privacy risks. Additionally, the availability of more training data for creating a voice model is likely to improve the accuracy of the synthesized voice. For example, the synthesized audio used in our user study were generated from voice models, with each model built using 40 short audio samples. As stated by the Lyrebird.ai voice synthesis service, providing more training samples improves the quality of the voice model and the synthesized speech.

6.7.2 Practical Deployment

A major goal of our proposed audio sanitizer is to improve the privacy of users without requiring any support from various transcription services. Our defense requires only the manipulation of audio on client devices before it is sent to the remote transcription services. Any device that accepts voice commands and does not perform speech recognition locally can leverage the audio sanitizer. To be effective, the audio sanitizer has to be placed somewhere in the path between the user and the transcription service so that it can intercept and sanitize the audio that comprises the voice command.

One possible point of interception is the communication link between the client device and the remote service. The audio sanitizer can capture the voice command from network packets and then forward it to the service after sanitization.
However, the communication between the client device and the remote transcription service usually happens over an encrypted channel\(^3\).

A more practical point of interception of audio data is within the client device itself, after the audio has been recorded by the microphone(s) and before it leaves the device. Relatedly, mechanisms for tracking and intercepting sensor data before delivering it to applications have previously been explored [129, 167]. For example, Xu and Zhu [167] propose a framework for Android smartphones that allows users to control the data generated by various sensors based on user-defined policies for the requesting application before forwarding the sensor data to that application. In particular, for audio data recorded by a microphone, their approach allows replacement of actual audio with mock data or with the addition of random noise. Thus, we can leverage such existing mechanisms on devices running Android to allow the audio sanitizer to intercept and sanitize the audio from the microphone before it is delivered to the application.

For in-home assistants with dedicated hardware such as Amazon Echo or Google Home, our defense will require support from the respective vendors. An ideal scenario would be to allow the user to run custom software on these devices. That way, we can directly integrate the audio sanitizer on such devices. A motivating example is the Amazon Echo that runs FireOS, which is an Android-based operating system [16, 32] and thus, can possibly use the same strategy as other Android devices.

\(^3\)We verified that Google Home and Amazon Echo use encrypted TLS connection to send voice commands to remote servers.
6.7.3 Additional Benefits

In addition to thwarting an adversary's attempt to build an accurate voice model of targeted speaker, the audio sanitizer also allow service providers to make stronger claims about user privacy. Current devices such as Amazon Echo and Google Home record and transmit any sound they hear after the activation word. Thus, any accidental triggering of such always-on voice assistants during confidential conversations poses a significant threat to user privacy [25]. As highlighted by recent events, governments can subpoena service providers for any such recordings [14, 24], which can harm a provider's efforts to alleviate public concern about the privacy risks of installing always-on listening devices. Service providers may opt to build audio sanitizers into their appliances and applications, as a way of assuaging privacy concerns.

6.8 Summary

The ubiquitous use of personal voice assistants has increased the opportunities to gain access to raw voice samples of the users. The advances in speech synthesis technologies have made it easier to create voice models from limited amount of raw speech data. Thus, the opportunity for a malicious attacker is ripe to leverage these advancements to spoof audio and perform voice spear phishing and various other forms of social engineering attacks.

In this chapter, we propose the audio sanitizer that reduces the amount of information in the voice commands leaving the client device. The audio sanitizer leverages the process of audio understanding in machines by allowing just enough information required for speech recognition to leave the client device and modifies the voice features that provide uniqueness to a speaker's voice. We show that audio
sanitizer significantly reduces the ability of an attacker to perform convincing voice spoofing attacks without having any significant impact of the accuracy of speech recognition services.
Chapter 7

Conclusion

This thesis demonstrates the feasibility of harnessing and exploiting the process of audio understanding in humans and machines and their differences.

In Chapter 3, we proposed the first hard-to-detect attacks against currently-deployed personal voice assistants. Our proposed attacks exploit the differences between the processes of audio understanding in humans and machines to create hidden voice commands. Our results indicate that such commands are practical and can be delivered to target devices even in the presence of background noise. Additionally, such hidden voice commands are unintelligible to humans listeners.

In Chapter 4, we discussed various potential defenses for securing voice assistants against our proposed attacks. We explored the benefits and limitations of various defenses including defenses that notify the users before executing voice commands and defenses that detect and prevent the execution of such attack commands. We showed that the differences in the process of audio understanding between hu-
man and machines can be used to filter out attack commands and only allow “good” commands to be executed by voice assistants.

In Chapter 5, we leveraged the process of audio understanding in humans to improve existing, state-of-the-art defenses against traffic re-identification attacks on encrypted VoIP streams. VoIP traffic re-identification attacks use the packet size distribution of the encrypted VoIP packets to infer attributes about the audio content. We proposed Whisper, the first unilateral defense to thwart such traffic re-identification attacks without requiring any participation from the receiver in a VoIP conversation. Whisper influences the size of encrypted packets by overlaying audio, inaudible to humans, to the actual audio message before it is encoded by a VBR codec. Our results show that Whisper significantly reduces an attacker’s ability to perform re-identification attacks while preserving the bandwidth savings and audio quality provided by the VBR codec.

Finally, in Chapter 6, we explored the threat posed by the advances in speech synthesis and the ubiquitous use of personal voice assistants to the public’s susceptibility to voice synthesis attacks. We showed that using available speech synthesis tools, an attacker with access to a small number of raw speech samples of the target can easily generate convincing spoofed audio in the target’s voice. To minimize access to raw speech samples from audio commands, we proposed the audio sanitizer for locally sanitizing voice inputs before they are transmitted to the cloud for processing. The audio sanitizer extracts audio information from voice commands that is relevant for speech recognition while stripping out information that significantly hinders the task of speech synthesis. Our results indicate that the audio sanitizer successfully prevents the attacker from generating convincing spoofed audio from sanitized audio commands without affecting the usability of personal voice assistants.
The work presented in this thesis is my own. Parts of Chapters 2–6 are reproductions of my previous publications [65, 150, 151, 152].
Chapter 8

References


[89] P. Hancke. webrtcH4cKS: - What’s up with WhatsApp and WebRTC? 

[90] H. E. Heffner and R. S. Heffner. Hearing Ranges of Laboratory Animals. In 

(USENIX), 2014.

speech quality model. In EURASIP Journal on Audio, Speech, and Music 
Processing, 2015.

acoustic modeling in speech recognition: The shared views of four research 

ing Technologies Symposium (PETS), 2003.

Perspective API Built for Detecting Toxic Comments. In arXiv preprint 


