Adversarial Command Detection Using Parallel Speech Recognition Systems

Peng Cheng[1], Arun Sankar M S[2], Ibrahim Ethem Bagci[3], and Utz Roedig[2]

[1] School of Cyber Science and Technology, Zhejiang University, Hangzhou, China
Key Laboratory of Blockchain and Cyberspace Governance of Zhejiang Province
peng_cheng@zju.edu.cn
[2] School of Computer Science and Information Technology, University College Cork, Cork, Ireland
a.sankar@cs.ucc.ie,u.roedig@ucc.ie
[3] VMware Inc., London, United Kingdom
bagcie@vmware.com

Abstract. Personal Voice Assistants (PVAs) such as Apple's Siri, Amazon's Alexa and Google Home are now commonplace. PVAs are susceptible to adversarial commands; an attacker is able to modify an audio signal such that humans do not notice this modification but the Speech Recognition (SR) will recognise a command of the attacker's choice. In this paper we describe a defence method against such adversarial commands. By using a second SR in parallel to the main SR of the PVA it is possible to detect adversarial commands. It is difficult for an attacker to craft an adversarial command that is able to force two different SR into recognising the adversarial command while ensuring inaudibility. We demonstrate the feasibility of this defence mechanism for practical setups. For instance, our evaluation shows that such system can be tuned to detect 50% of adversarial commands while not impacting on normal PVA use.

1 Introduction

Personal Voice Assistants (PVAs) such as Apple's Siri, Amazon's Alexa and Google Home are now commonplace. A PVA can be integrated as functionality in other devices such as smart phones or TVs or may be implemented as dedicated device referred to as smart speaker. We use PVAs to interact with infrastructures such as our smart home and services such as e-mails and news.

There are a number of PVA security and privacy concerns and research has investigated a large variety of attacks on these systems. One prominent attack example is the so called *hidden command injection*. The aim of such attack is to supply a specially crafted voice signal, referred to as adversarial command, to the PVA which is interpreted differently by the PVA than it is by humans. For example, the supplied adversarial command may be interpreted by humans as *'Alexa, tell me what the weather is like'* while the SR of the PVA interprets this signal as *'Alexa, open the front door'*. An adversarial command is created by

adding small perturbations to an audio recording until the PVA's SR recognises the intended command of the attacker instead of the command contained in the original audio recording. If the perturbations are small and added carefully, a human will not notice the modification of the audio signal while the SR algorithms recognise different words. How to create adversarial commands has been studied in detail [4] [11]. However, less effort has been put into devising defence methods against this serious attack form.

In this paper we describe a low complex defence method against adversarial attacks based on the weak transferability of adversarial commands. The generation of an adversarial command that can successfully target multiple SR systems is still an open question [15]. Our method makes use of a second SR, we call it the *protection SR*, within a PVA which analyses the supplied voice sample in parallel to the *main SR*. The speech transcription output of the protection SR is compared with the transcription output of the main SR and only if both outputs are a close enough match the transcription output is accepted and the command is executed. The protection SR may use different training data or even an entire different SR architecture compared to the main SR.

The protection SR does not have to produce the same transcription quality as the main SR. Voice recognition of this component must only be sufficiently accurate to provide protection, transcription accuracy is delivered by the main SR. Thus, the protection SR can be simpler and can also be based on much smaller training data. It is possible to implement the protection SR without much resource requirements and it is possible to use frequent re-training. Frequent retraining adds additional complexity for a potential attacker that may try to craft an adversarial command targeting main and protection SR jointly. It is assumed to be infeasible for an attacker to add unnoticeable perturbations to the original audio such that two entirely different SR are tricked into producing the same transcriptions. The main contributions of this paper are:

- *Adversarial Command Detection (ACD):* We describe a novel protection mechanism against adversarial commands using parallel SR systems.
- *Demonstration of ACD:* We demonstrate the effectiveness of ACD using 20 adversarial commands and show that our ACD using Pocketsphinx [7] and Kaldi can detect all adversarial commands. We also show that the ACD does not prevent normal PVA operations due to false positives.
- *ACD Complexity:* We show that the protection SR can be significantly less complex than the main SR in terms of architecture and training data. Thus, frequent retraining of the protection SR is feasible, providing a ACD as moving target defence.

The remaining paper is structured as follows. Section 2 provides a very brief introduction to Automatic Speech Recognition (ASR) and describes adversarial command generation. Section 3 discusses related work and Section 4 introduces our novel Adversarial Command Detection (ACD) method. In Section 5, Section 6 and Section 7 we describe our evaluation setup, experimental results and discussion. Section 8 concludes the paper.

2 Preliminaries

In this section we give a brief definition of a PVA as considered in this work. We also provide a definition of adversarial commands and provide a description on how these are crafted.

2.1 Personal Voice Assistant (PVA)

A PVA is a service which understands voice commands and is able to take corresponding actions. A PVA may reuse hardware of existing devices such as mobile phones or TVs or may use dedicated hardware such as a smart speaker.

The acoustic signal (i.e. human voice) is captured by microphones. Usually, the signal is processed locally to identify a wake word (e.g., 'Alexa' or 'Hey Google'). For wake word recognition a simple SR system is sufficient. After the wake word is recognised the following audio recording is transported to a back-end where a more sophisticated SR system analyses the audio sample to extract the command. After command extraction the back end system initiates the required action (interact with a system or query a service). User feedback in form of audio may be generated and transported to the local device where it is played back via speakers.

In this work we assume one local SR component is used to implement a PVA and we do not distinguish wake word recognition SR and back-end SR. However, our work can be applied to systems that distribute SR.

2.2 Hidden Commands

Hidden voice command injection aims to inject voice commands into a PVA without users noticing this injection. The injected command is 'hidden' from users present in the vicinity of the PVA. In order to conceal this interaction existing work has looked at various techniques ensuring that a person is unable to hear the submitted command while the PVA's ASR is able to understand it. While these techniques are the essential component to enable hidden voice commands it is also often necessary for an attacker to modify other elements of PVA interaction. After submitting a command, the PVA usually responds with a confirmation via it's speakers. For example, the voice command for a home automation system 'Alexa, open the front door' would result in a response 'Front door opened' which an attacker would need to suppress too in order to achieve a fully hidden interaction. However, it has to be noted that not all services provide a user with feedback and in some cases a user may simply ignore unexpected feedback.

Three types of hidden commands have to be distinguished: *Hardware Non-Linearity*, *Obfuscated Commands* and *Adversarial Commands*.

Work in the first category targets the analogue signal processing path of a PVA and makes use of the fact that humans are unable to hear in the high frequency range (typically above 18kHz). The voice command is submitted in

the frequency space unnoticeable to users while non-linear behaviours of the analogue signal processing path ensures that the signal is processed by SR.

The second class of work aims at submission of an audio signal which humans perceive as noise, the command is understood by PVAs but not by humans. For this purpose, the attacker starts with the target command and this audio signal is gradually changed until it becomes unintelligible for a human but the PVA still decodes the command. The resulting audio signal is called the obfuscated command.

The third class is similar to the second. The original audio signal (original command) is gradually modified until the PVA recognises the target command while a human still hears the original command. The resulting audio signal is called the adversarial command.

In this paper we focus on methods for hidden command detection of the third type: adversarial commands. We focus on this specific type as it is considered the most effective attack and consequently attracts currently most research effort. However, our proposed defence method may also protect against the other two types but we have not verified this in our experimentation.

2.3 Obfuscated and Adversarial Commands

The purpose of ASR is to transcribe speech to corresponding text. This process can be defined as:

$$y = \arg\max_{\widetilde{y}} p(\widetilde{y}|x) \tag{1}$$

x here is the audio input, and $\tilde{y}$ are all possible transcription candidates. The ASR aims to find the most likely transcription y given the audio input x. Once the ASR has been trained it's function is $y = f(x)$.

A human listening to the audio signal x also interprets the signal and normally would conclude that the same transcription y recognised by the ASR is the meaning of the command. This process can be described as $y = f_H(x)$ with f_H describing the human's processing capability.

An adversary can modify an input signal x by adding perturbation δ, resulting in $x' = x + \delta$. The following situation may arise when an ASR decodes x':

$$y = f(x') \quad \text{and} \quad \emptyset = f_H(x') \tag{2}$$

y here is the obfuscated command transcription which remains the same as the one decoded from unperturbed input x. However, a human may not perceive the same transcription y this time from the audio signal x' (it is perceived as noise; $f_H(x') = \emptyset$ which means the human transcription is empty). In this case, the audio input x' is called the *obfuscated command*.

There is as well the other situation where $y = f_H(x')$ and $\emptyset = f(x')$ which means the ASR is unable to transcribe the input while a human is understanding the command well. There is work in this direction (such as work by Abdullah et al. [3]) which aims to prevent machines listening into conversations.

$$\emptyset = f(x') \quad \text{and} \quad y = f_H(x') \tag{3}$$

The situation of interest in this paper is where $y = f_H(x')$ and $y' = f(x')$ which means the ASR transcription and human transcription are different. In this case x' is called an *adversarial command*:

$$y' = f(x') \quad \text{and} \quad y = f_H(x') \tag{4}$$

In case of the adversarial command, even with the added perturbation, a human still perceives the adversarial audio input x' as original benign command transcription y, while an ASR recognises the audio input x' as the adversarial command transcription y'.

We distinguish so called *targeted* and *non-targeted* adversarial commands. In case of a *targeted* adversarial command the attacker is interested in one specific command transcription T which is carefully selected ($y' = T$). In case of a *non-targeted* adversarial command the attacker does not care about what specific command would be decoded by the ASR; the attacker only wants to ensure that human and machine transcription are not the same.

2.4 Adversarial Command Generation

To create an adversarial command it is helpful for the attacker to have access to the internal workings of the ASR. An attack relying on such internal knowledge (e.g. such as the trained Deep Neural Network (DNN) model) is referred to as a *white-box* attack. If the attacker is not able to access the internals and is only able to obtain ASR decoding results the attack is classified as a *black-box* attack. Generally, attacks assuming the ASR as a black-box are more difficult to execute and have a lower attack performance (i.e. successfully generating adversarial examples providing the desired transcription). It has to be noted that we generally have to assume that an attacker has access to the ASR and a white-box attack is likely. As in any other area of computer security we cannot provide security by obscurity and assume that the ASR remains hidden.

The exact process of generating adversarial commands may vary depending on the ASR model, black-box/white-box assumption and perturbation target such as feature vectors or raw audio input. Recent work focuses on adding perturbations directly to the audio input rather than the result of the preprocessing (e.g., FBANK) as this approach reduces the perceptible noise in adversarial examples [4].

We use the generation of adversarial commands for a Deep Neural Network - Hidden Markov Model (DNN-HMM) ASR as example. Adversarial commands are generated through an iterative process. In each iteration, the output of the DNN (the acoustic model) is compared with the target using a loss function. Then the gradient of the loss function with respect to the corresponding input is calculated through back-propagation. By finding the perturbed input resulting in the local/global minimum it is ensured that the input is transcribed as the target

command. In addition, the perturbation value is constrained by a threshold, ensuring that people cannot perceive the difference between the new signal and the original audio input. There are variations in different studies in regard to techniques on where to add the perturbations. For example, they can be added to the feature vectors such as Mel-Frequency Cepstrum Coefficient (MFCC) or directly to the raw audio input.

3 Related Work

Recent work has thoroughly investigated the construction and efficiency of adversarial examples. Among the earlier works, adversarial attacks are generated using tuned MFCC features [11] and inverse feature extraction [5] in the form of obfuscated commands that are not intelligible to human listeners. By exploiting the temporal and frequency masking property of the psychoacoustic model, the voice commands are embedded in speech or music which is perceived by listeners as speech or music but recognized by ASR as commands [14] [6] [10]. In this work, we used the method proposed by Schönherr et al. [10] that hides voice commands in audio locations which minimizes perceptual distortion.

Only a few works investigating the construction of adversarial examples for ASR have also analysed in detail if the examples are transferable. CommanderSong [14] that targets Kaldi ASR has found to be unsuccessful on DeepSpeech but an improvement for this is obtained in [4] by generating a few adversarial examples that targets DeepSpeech using the adversarial commands for Kaldi ASRs. Similarly, the adversarial examples that are successful on DeepSpeech2 with little perceptual distortion generate invalid transcription for Google Voice [6]. Abdullah et al. [3] obtained a reasonably good transferability for an *evasion* attack. A systematic approach for the generation of transferable adversarial commands is not yet achieved.

Some existing work has proposed defence mechanisms against adversarial examples. This includes a neural network-based classifier for detection of hidden commands [5] and the use of audio pre-processing methods (addition of noise, down sampling, audio compression, band-pass filtering, audio panning) in individual or in combination to render adversarial examples ineffective [14] [9]. The temporal consistency of speech signal is exploited in [13] for developing the countermeasure due to the limited robustness shown by signal processing methods towards adversarial attacks [12]. The narrow-band vocoder G.729 along with Pulse-code Modulation (PCM) is used to eliminate the perturbations due to adversarial commands [8]. The ASR outputs of this filtered audio signal and the raw input signal are compared to detect adversarial commands when the difference is greater than a threshold.

Closest to our work is work by Zeng et al. [15] which proposes a multiversion programming inspired approach to detect audio adversarial examples. This work also uses additional ASRs to detect adversarial examples. The proposed detection mechanism consists of one main ASR system based on DeepSpeech and three auxiliary ASR systems that comprise DeepSpeech, Long Short-Term Memory

(LSTM) based Google Cloud Speech, and Amazon Transcribe with unknown internal architecture. The detection accuracy of this approach is very high but it drops with the similarity between the commands and transcribes. In addition to the internal architecture, the type and volume of training data is also a significant factor that makes any ASR system unique. However, our work differs as we aim to minimise complexity of the defence mechanism. Reduced complexity is essential to support implementation on resource constraint PVAs or to facilitate deployment on scale when used in a cloud infrastructure. Our work also differs as we investigate not only the effectiveness of different ASR architectures but also the impact of using a reduced training data sets.

4 Adversarial Command Detection (ACD)

4.1 Threat Model

The attacker may have access to a PVA's SR when crafting adversarial commands. In addition, an attacker may also have access to the protection SR which we propose as defence method.

A1: General Attacks: We assume the attacker is only able to inject commands via the audio channel. There is no way for the attacker to bypass the SR entirely; i.e. by a conventional hack of the PVA.

A2: Adversarial Command: We assume that the attacker is only able to supply a rogue command as a hidden command constructed as adversarial command. We assume that the attacker must submit a command within an audio sample such that a present user is not aware of this embedded threat. We do not consider direct non-authorised interaction with the PVA or submission of hidden commands using different techniques.

A3: Main SR Access: We assume that the attacker has access to the main ASR; i.e. we consider a white-box attack. The attacker has full access to the main ASR when crafting the attack signal. This is a reasonable assumption as it is not feasible to keep an ASR used for millions of devices a secret.

A4: Protection SR Access: We assume that the attacker does not have access to the protection ASR; i.e. we assume a black-box attack by considering the protection ASR as a moving target. As we will show, it is possible to frequently retrain the protection ASR which ensures that an attacker is not able to obtain a copy of the used protection ASR. It has to be noted that there is currently no study showing that it is feasible to construct an adversarial command targeting multiple ASR in parallel under either white or black-box assumption. However, making the assumption that the protection ASR is black-box will make this problem significantly harder as the attacker would need to craft an adversarial command working with all possible ASR configurations at the same time.

4.2 ACD Approach

The ACD approach is shown in Figure 1. The *main ASR* of the PVA is accompanied by a *protection ASR* and both process the incoming audio signal. Both

ASR may share the same front-end (Microphone, filters, gain control, ...) and both ASR produce a transcription of voice. The main ASR transcription is only passed on to the PVA command execution if a comparison of both transcriptions determines that no adversarial command is present.

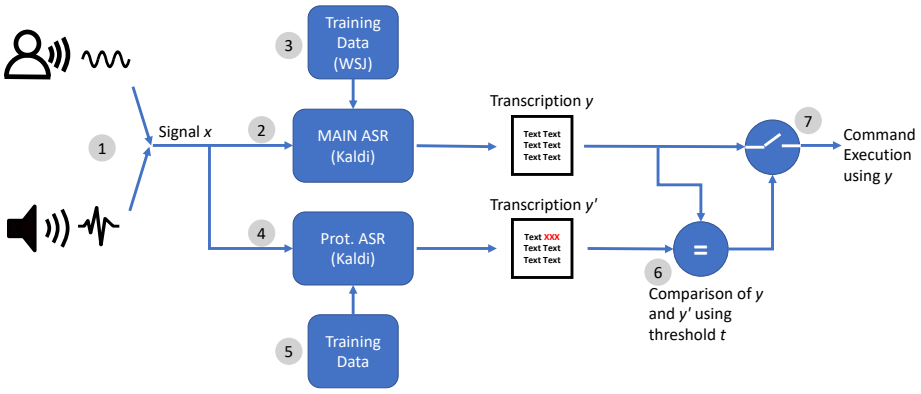

Fig. 1: Adversarial Command Detection (ACD) - The audio signal is analysed by the main ASR and the protection ASR in parallel. If both transcriptions differ significantly, defined by a threshold, the command is rejected.

1. An input signal x is provided. Either the audio input signal is provided by a human speaker or it may be supplied by an attacker via a loudspeaker. In case of an attacker the supplied signal is the adversarial command.
2. The voice audio signal x is fed to the main ASR used to transcribe the voice signal into text y.
3. The main ASR is a sophisticated ASR, using a complex structure and trained with a large corpus to provide an accurate transcription y for a diverse set of speakers. The transcription process is described as: $y = f_M(x)$.
4. A copy of the audio signal is also fed to a protection ASR which creates it's own text transcription y'.
5. The protection ASR is far less sophisticated than the main ASR and is also trained with a much smaller data set. The transcription is less accurate than that of the main ASR. The protection ASR produces the following transcription output: $y' = f_P(x)$.
6. The output of both ASRs y' and y is compared using the metric Word Error Rates (WER) (WER is introduced in Section 5) against an error threshold

t. The WER difference between y' and y should not be greater than t. If the input is a legitimate voice command from a user, the difference between the two transcription is assumed to not be greater than the threshold. The threshold can be selected according to goals of the overall system.
7. Only if the recognition difference is below threshold t the transcribed command is considered valid and transcription y is passed to the PVA command execution.

4.3 ACD and Protection ASR Properties

To create an adversarial command the attacker executes an iterative process in which the signal is modified by adding inaudible perturbations such that the ASR recognizes the desired command. The signal is adapted taking into account the ASR's model which is defined by (i) ASR architecture and (ii) ASR training data. We assume that if the protection ASR model differs from the main ASR it is infeasible for an attacker to modify iterative the input signal such that two entirely different ASR are forced to produce the same transcription while ensuring that signal modifications remain inaudible.

Different Architectures Different ASR architectures may use different types of features and may also extract different internal features. The adversarial example generation is an iterative process which aims to find the optima for the loss function with respect to the input by adding perturbation. It will be difficult for an attacker to add perturbations with the aim of changing features important for one architecture and also the other.

Different Training Data The details of the model parameters used by the ASRs depend on the used training data. The parameter values keep being modified to improve prediction results during the training process. Thus, when generating an adversarial command the attacker needs to take into account not only the ASR architecture but also the specific trained model. An adversarial command generated for an ASR is only likely to work when the same trained model is used.

5 Evaluation Setup

The ACD is evaluated using a number of different ASRs in the role of a protection ASR. A number of benign and adversarial commands are used to evaluate ACD detection capabilities and normal operation scenarios.

5.1 ASR Selection

Our selection of ASRs used for evaluation is summarised in Table 1. Each ASR model (comprising architecture and model) is given a label which we use in the remaining document for reference.

ASR Label	ASR Variant	ASR Architecture	Training Data
MASR1	Kaldi nnet2	DNN-HMM	WSJ
PASR1	Pocketsphinx	GMM-HMM	Unknown1
PASR2	Kaldi nnet3	DNN-HMM	Unknown2
PASR3	Kaldi GMM	GMM-HMM	WSJ
PASR4	Kaldi GMM	GMM-HMM	50% of WSJ

Table 1: Kaldi using an nnet2 model is used as main ASR (MASR1). Three different ASR are used as protection ASRs, labeled PASR1, PASR2, PASR3 and PASR4. The protection ASRs are Pocketsphinx and different Kaldi variations.

MASR1 As main ASR (Label *MASR1*) we use an nnet2 Kaldi model which is a DNN-HMM structure using the Wall Street Journal (WSJ) corpus as training data. Note that the latest Kaldi is an nnet3 chain model. However, we use a nnet2 Kaldi as the main ASR as we use adversarial command generation based on work by Schönherr et al. [10] which relies on this ASR variant.

The nnet2 Kaldi used by Schönherr makes use of some modifications. The feature extraction and the DNN acoustic model are combined. This integration is for the convenience of adding perturbation directly to the input rather than the intermediate features when generating adversarial commands. According to Schönherr, this design modification does not affect the accuracy of the ASR system. We treat this modified nnet2 Kaldi model and the standard one as equivalent in this work. When evaluating adversarial commands we use this modified nnet2 Kaldi ASR; when evaluating benign commands we use the standard nnet2 Kaldi ASR.

PASR1 The first candidate of a protection ASR (Label *PASR1*) is the Pocketsphinx for a standard Raspberry Pi 3 Model B+. Pocketsphinx is using a Gaussian Mixture Model - Hidden Markov Model (GMM-HMM) model, while the the main ASR MASR1 is using DNN-HMM model. These two models have completely different acoustic model architectures. Although it is not clear what corpus is used to train Pocketsphinx, it is safe to assume Pocketsphinx is trained with different training corpus than the main ASR (Pocketshinx is provided with the already trained model and a clear description of the used training data is not provided). GMM-HMM training requires less resources and is considered to be an older fashion of ASR compared to DNN-HMM model. However, as the protection component can handle a lower transcription accuracy for the benefit of less complexity Pocketsphinx is a suitable choice.

PASR2 The second candidate (Label *PASR2*) is from an open-source project called Zamia Speech [2] which provides pre-built Kaldi ASR packages for Raspbian (A commonly used Operating System (OS) for Raspberry Pi) complete with pre-trained models for English. It uses Kaldi nnet3 chain audio models. nnet3 and nnet2 are both DNN, but nnet3 supports more general networks. Therefore, we treat nnet3 as a variation of the Kaldi DNN. Specifically, we use

$kaldi-generic-en-tdnn\_f$ which is a pre-trained nnet3 chain model trained on 1200 hours of audio. We treat it as an ASR with different architecture and trained with different training dataset compared to the main ASR.

PASR3 The third candidate (Label *PASR3*) is the standard GMM-HMM model from Kaldi trained using the the Wall Street Journal (WSJ) Corpus. Note that although the principal architecture of this ASR is the same as for Pocketsphinx, but the specific parameters are different. Note this candidate is trained using the same dataset as the main one, but it has a completely different architecture.

PASR4 The fourth candidate (Label *PASR4*) is identical to PASR3 except the used training data. Only 50% of the Wall Street Journal (WSJ) Corpus are used for training the ASR.

In summary, MASR1 uses a DNN-HMM architecture using the WSJ corpus as training data. PASR1 uses a different architecture and different training data compared to MASR1. PASR2 shares the architecture with MASR1 but uses different training data. PASR3 shares the training data with MASR1 but uses a different architecture. PASR4 uses a less complex training data set compared to PASR3.

5.2 Adversarial and Benign Command Generation

Adversarial Commands We generate the adversarial commands based on work by Schönherr et al. [10]. 20 adversarial commands are generated; the commands are hidden in 20 music segments as provided on the GitHub repository [1].

Benign Commands To show how the main ASR and the defence ASRs perform in a normal setting we also generate benign commands. We generate 20 benign commands based on the transcriptions of the 20 adversarial commands using the Google online Text-to-Speech (TTS).

5.3 Experiment Setup

We conduct three sets of experiments. For each set we used the main ASR MASR1 and one of the three protection ASRs PASR1 to PASR3. In each experimental set we evaluate how the 20 adversarial commands and the 20 benign commands are classified by the ACD. The audio commands are directly fed into the main ASR and the protection ASR candidates.

For each adversarial and benign command, we compare the decoding results between the main ASR and the protection ASR using the WER metric. The ACD decision in dependency of WER threshold t is recorded. Based on the ground truth we record if this was a true positive (TP), false positive (FP), true negative (TN) or false negative (FN) decision.

True positive means that we decide the command is an adversarial one and the decision is correct; false positive means we decide the command is an adversarial one and it turns out the command is benign; true negative means we

decide the command is a benign one and this decision is correct; false negative means we decide the command is a benign one but actually it turns out to be adversarial.

5.4 Evaluation Metrics

Word Error Rates (WER) WER is defined as

$$WER = ((N_{sub} + N_{ins} + N_{del}))/N_{ref} \tag{5}$$

where N_{sub} is the number of words which are incorrectly transcribed, N_{ins} is the number of words which appear in the current transcription but are not present in the reference, and N_{del} is the number of words in the reference that do not appear in the transcription. Note that WER can be greater than 100% as the transcription can be longer than the reference.

Receiver Operating Characteristic (ROC) We draw ROC curves for the ACD with the False Positive Rate (FPR) as the x-axis and the True Positive Rate (TPR) as the y-axis. Each ROC curves shows FPR versus TPR for all possible ACD decision thresholds t. TPR is defined as:

$$TPR(t) = TP(t)/(TP(t) + FN(t)) \tag{6}$$

FPR is defined as:

$$FPR(t) = FP(t)/(FP(t) + TN(t)) \tag{7}$$

For each ROC curve, the area under the curve (AUC) is calculated and the ACD has a better prediction skill the greater the AUC value is. An ACD with no skill has an AUC of 0.5 and a useful ACD mut provide an AUC value above 0.5.

6 Evaluation Results

We first present a performance evaluation of the five different ASR used (MASR1, PASR1, PASR2, PASR3, PASR4 as shown in Table 1). Each of the ASR are used to decode 20 benign and 20 adversarial commands. Then we evaluate the ACD performance where different combinations of main ASR and protection ASR are used.

6.1 Decoding Results of Normal Speech

The 20 benign commands are fed to the different ASR; the results are shown in Table 2 and there exists always a certain amount of WER. From a speech recognition perspective, the WER has to be minimized and there exists various methods for doing it. This include the following Natural Language Processing (NLP)

ASR	ASR Variants	ASR Architecture	WER Begnin	WER Adverserial
MASR1	Kaldi nnet2	DNN-HMM	39.73%	12.33%
PASR1	Pocketsphinx	GMM-HMM	63.01%	139.73%
PASR2	Kaldi nnet3	DNN-HMM	73.28%	98.63%
PASR3	Kaldi GMM	GMM-HMM	44.52%	97.26%
PASR4	Kaldi GMM	GMM-HMM	47.94%	100%

Table 2: The effect of the 20 benign TTS and 20 adversarial commands on variations of ASR systems. The benign commands are best recognised by MASR1 which is based on kaldi nnet2 model. The adversarial commands are only effective on MASR1, the ASR for which the commands were generated. For all other ASR the WER is high, indicating an unsuccessful transcription.

component to analyse intents and semantics of the ASR transcription which can mitigate some errors generated in the ASR decoding step. Since our goal is not to optimize parameters of ASR to achieve high performance but to verify the proposed defence method, the main concern is in the relative variation of WER with benign and adversarial commands. Also the reasons for the performance variations for the ASR models are analysed in Section 7. MASR1 based on the advanced DNN-HMM architecture shows the best transcription results for benign TTS commands. Even with same architecture used in PASR3 and PASR4, the transcription performance of PASR4 is lower than PASR3 as it uses only half the training data. The worst performance in decoding commands is shown by PASR2 followed by PASR1 that uses different corpus from WSJ which is used for training the rest of ASRs.

MASR1 First we use the nnet2 Kaldi (main ASR). Using this ASR results in WER of 39.73% with 31 insertions, 17 deletions and 87 substitutions.

PASR1 The Pocketsphinx decoding results of these 20 human spoken commands result in a WER of 63.01% with 48 insertions, 12 deletions and 62 substitutions.

PASR2 The decoding results from Kaldi nnet3 compared to the ground truth transcription result in a WER of 73.28% with 3 insertions, 50 deletions and 41 substitutions.

PASR3 Feeding these commands to the Kaldi GMM-HMM model results in WER of 44.52% with 11 insertions, 11 deletions and 55 substitutions.

PASR4 The decoding results for the 20 human spoken commands are 47.94% with 20 insertions, 8 deletions and 5 substitutions.

6.2 Decoding Results of Adversarial Commands

The 20 adversarial commands are fed to the different ASR. As shown in Table 2, it is clear that the crafted adversarial commands are only effective against the ASR used in the adversarial command generation. Any other ASR, differing in architecture, training data or both does not transcribe the commands usefully. The detailed results are as follows:

MASR1 The adversarial commands are successful resulting in the best overall WER of 12.33%. Specifically, 2 word insertions, 9 word deletions and 7 word substitutions are recorded when comparing the transcription with the reference. The accuracy is 35% which means 7 sentences out of 20 are exactly the same as the target transcription (all words in the sentence are correct). This proves the white-box attack is successful as expected. The adversarial commands are crafted specifically for MASR1.

PASR1 The decoding results from Pocketsphinx is far from the target, resulting in WER of 139.73% when compared with the reference text. Specifically, 64 word insertions, 11 word deletions and 129 word substitutions.

PASR2 When feeding the adversarial commands to the Kaldi nnet3 chain model runnig on a raspberry Pi, none of the target sentences are correctly transcribed. Specifically, WER is 98.63% with 0 insertion, 129 deletions and 15 substitutions.

PASR3 None of the target sentences are correctly transcribed. Specifically, WER is 97.26% with 3 insertions, 101 deletions and 38 substitutions.

PASR4 The results are similar to PASR3 with a reduced training data. Specifically, WER is 100% with 1 insertion, 118 deletions and 27 substitutions.

6.3 Adversarial Command Detection (ACD)

We evaluate the ACD ASR combinations. The benign commands and adversarial commands are fed to the system. For each combination, we evaluate the WER threshold and draw the ROC curve (Figure 2).

A good ACD should produce a ROC curve passing close to the top left corner (i.e. from (0,0) via point (0,1) to (1,1)). A curve following a diagonal (i.e from (0,0) to (1,1) would represent a bad ACD that cannot discriminate and represents a random guess. A good ACD would have an AUC value close to 1 while a bad ACD would have an AUC value close to 0.5.

As shown in Figure 2, the four pairs of ASR do not differ significantly in terms of detection performance. However, the ACD using PASR1 (Pocketsphinx) provides the best ROC curve. The AUC value for PASR1 is $AUC_{PASR1} = 0.898$ which is the highest among the four ($AUC_{PASR2} = 0.833$, $AUC_{PASR3} = 0.889$, and $AUC_{PASR4} = 0.773$).

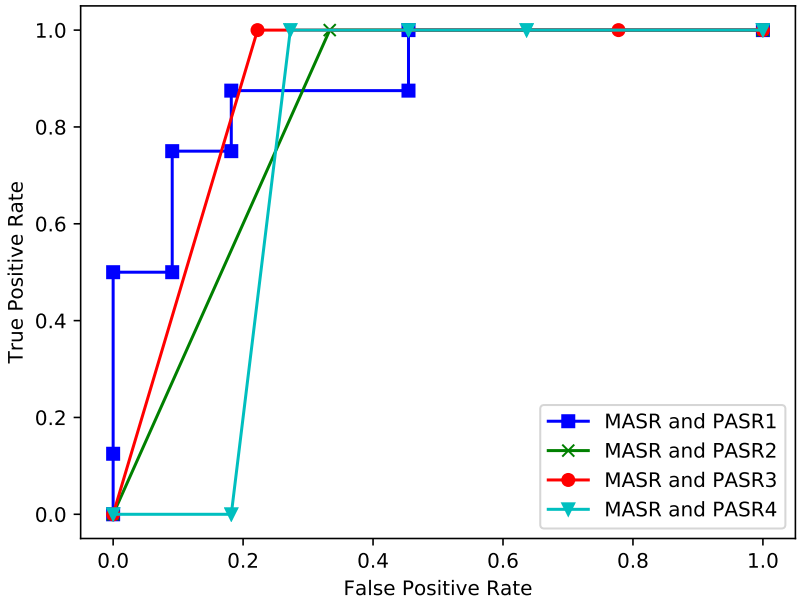

Fig. 2: ROC curves for all protection ASR options. Pocketsphinx (PASR1) provides the best option for the ACD with an AUC value of $AUC_{PASR1} = 0.898$.

PASR1 We see that Pocketsphinx (PASR1) tends to transcribe our 20 music based adversarial commands to longer sentences than the original command transcription, which results in WER values above 100%. This makes it easier to select a threshold with higher TPR and lower FPR.

PASR2 The Kaldi nnet3 (PASR2) model often only transcribes a few words or even nothing when fed with our 20 adversarial commands. This results in a WER value of 98.63% which is greater than the WER in recognizing the normal TTS generated commands by a value of only 25.35. This smaller WER difference is due to the worst decoding performance shown by Kaldi nnet3 in recognising spoken commands which makes it not an optimal choice when coupled with the main ASR here in our setup.

PASR3 and PASR4 The Kaldi GMM-HMM (PASR4) uses half the training data than the Kaldi GMM-HMM (PASR3). These two models have similar performances in decoding the 20 adversarial commands. The decoding results for most samples are either shorter than the target transcriptions or nothing can be decoded. There are only two WER results beyond 100% for both PASR3 and PASR4. Overall, the WERs for both PASR3 and PASR4 decoding 20 adversarial commands are high and around 100%, so the true positive value for both

of these reach 100% easily. When decoding TTS generated benign commands, the performance of PASR3 is slightly better than PASR4 as shown in Table 2. There are more high WER values for PASR4 than PASR3, so as the threshold varies, the false positive rate of PASR4 is greater than that of PASR3, which is presented in Figure 2.

7 Discussion

7.1 Observations

The experiments show that any ASR that differs in architecture and/or training data is usable as protection ASR. In our experimentation setup, Pocketsphinx (PASR1) turned out to be most effective but the other candidates are usable too. Depending on the application scenario, different thresholds might be used. For example, an $FPR = 0$ can be chosen while still obtaining an $TPR = 0.5$ in case of PASR1. For such setting 50% of the time an attack will be detected while not preventing normal use of the system.

Different Architectures As the ACD mechanism should be integrated in a PVA ecosystem it is important to consider resource requirements of the protection mechanism. Using this protection, voice is analysed by two ASR components instead of one. The ACD implementation may run on a dedicated device (e.g. a smart speaker or phone) or within a cloud-based back-end infrastructure. In either case, additional resource use of the ACD should be limited. Hence, it would be desirable to use a much less resource intensive ASR as protection ASR. The experiments show that this approach is feasible. For example, Pocketsphinx is an ASR designed for systems with limited resources and is less complex than Kaldi which we used as main ASR.

Different Training Data In order to increase difficulty for an attacker to bypass the ACD it should not be possible for the attacker to obtain knowledge on the internal workings of the protection ASR. This can be achieved by frequently changing the configuration of the ASR by using a different training data set. Thus, the protection ASR becomes a moving target. However, frequent retraining requires resources and such effort should be limited. The effort can be limited by reducing the training effort by reduction of the used training data. PASR4 uses half the training data compared to PASR3 producing comparable ACD protection results. In our setup this 50% reduction in training data led to a training time reduction of 22%.

7.2 Limitations

While our work shows the principle feasibility of ACD there are limitations which we would like to address in future work: *Adversarial Commands:* The adversarial commands are generated based on adding perturbations to music rather than

normal speech. *Benign Commands:* We used the Google online TTS service to generate the 20 benign commands. We would like to use human speakers for further evaluation. *Sample Size:* The number of used samples was relatively small.

8 Conclusion

We increasingly rely on PVA to interact with smart environments and services. It is essential that security of these systems can be ensured. Adversarial commands are a serious threat to PVAs. While it is well understood how adversarial commands can be generated and used little work has been carried out to devise defence methods.

In this paper we have shown that it is possible to use a parallel ASR (a protection ASR) to defend against adversarial commands. In our experiments it is shown that a less capable protection ASR is sufficient to achieve protection. It is possible to implement the protection ASR without much resource requirements and it is possible to use frequent re-training. Thus, an attacker needs to adapt to an ever changing target.

The efficacy of our proposed protection mechanism has been shown for the evaluated scenarios. However, it would be desirable to provide a formal proof that it is impossible for an attacker to construct a hidden command that can target two different ASR systems.

Acknowledgement

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant number 19/FFP/6775. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

References

1. Adversarial Attacks. https://github.com/rub-ksv/adversarialattacks, online; accessed 026-July-2020
2. Zamia Speech. https://github.com/gooofy/zamia-speech#asr-models, online; accessed 026-July-2020
3. Abdullah, H., Rahman, M.S., Garcia, W., Blue, L., Warren, K., Yadav, A.S., Shrimpton, T., Traynor, P.: Hear" no evil", see" kenansville": Efficient and transferable black-box attacks on speech recognition and voice identification systems. arXiv preprint arXiv:1910.05262 (2019)
4. Carlini, N., Wagner, D.: Audio adversarial examples: Targeted attacks on speech-to-text. In: 2018 IEEE Security and Privacy Workshops (SPW). pp. 1–7 (May 2018). https://doi.org/10.1109/SPW.2018.00009

5. Carlini, N., Mishra, P., Vaidya, T., Zhang, Y., Sherr, M., Shields, C., Wagner, D., Zhou, W.: Hidden voice commands. In: Proceedings of the 25th USENIX Security Symposium (USENIX Security'16). pp. 513–530. USENIX Association, Austin, TX (Aug 2016), https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/carlini
6. Cisse, M., Adi, Y., Neverova, N., Keshet, J.: Houdini: Fooling deep structured prediction models (2017)
7. Huggins-Daines, D., Kumar, M., Chan, A., Black, A.W., Ravishankar, M., Rudnicky, A.I.: Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In: Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on. vol. 1, pp. I–I. IEEE (2006)
8. Jiajie, Z., Zhang, B., Zhang, B.: Defending adversarial attacks on cloud-aided automatic speech recognition systems. pp. 23–31 (07 2019). https://doi.org/10.1145/3327962.3331456
9. Rajaratnam, K., Shah, K., Kalita, J.: Isolated and ensemble audio preprocessing methods for detecting adversarial examples against automatic speech recognition. In: Proceedings of the 30th Conference on Computational Linguistics and Speech Processing (ROCLING 2018). pp. 16–30. The Association for Computational Linguistics and Chinese Language Processing (ACLCLP), Hsinchu, Taiwan (Oct 2018), https://www.aclweb.org/anthology/O18-1002
10. Schönherr, L., Kohls, K., Zeiler, S., Holz, T., Kolossa, D.: Adversarial attacks against automatic speech recognition systems via psychoacoustic hiding. In: Proceedings of the 2019 Network and Distributed System Security Symposium (NDSS '19) (2019)
11. Vaidya, T., Zhang, Y., Sherr, M., Shields, C.: Cocaine noodles: Exploiting the gap between human and machine speech recognition. In: Proceedings of the 9th USENIX Workshop on Offensive Technologies (WOOT '15). USENIX Association, Washington, D.C. (Aug 2015), https://www.usenix.org/conference/woot15/workshop-program/presentation/vaidya
12. Yang, Z., Li, B., Chen, P., Song, D.: Characterizing audio adversarial examples using temporal dependency. CoRR **abs/1809.10875** (2018), http://arxiv.org/abs/1809.10875
13. Yang, Z., Li, B., Chen, P.Y., Song, D.: Towards mitigating audio adversarial perturbations (2018), https://openreview.net/forum?id=SyZ2nKJDz
14. Yuan, X., Chen, Y., Zhao, Y., Long, Y., Liu, X., Chen, K., Zhang, S., Huang, H., Wang, X., Gunter, C.A.: CommanderSong: A systematic approach for practical adversarial voice recognition. In: Proceedings of the 27th USENIX Security Symposium (USENIX Security '18). pp. 49–64. USENIX Association, Baltimore, MD (Aug 2018), https://www.usenix.org/conference/usenixsecurity18/presentation/yuan-xuejing
15. Zeng, Q., Su, J., Fu, C., Kayas, G., Luo, L., Du, X., Tan, C.C., Wu, J.: A multiversion programming inspired approach to detecting audio adversarial examples. In: 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 39–51 (2019). https://doi.org/10.1109/DSN.2019.00019