StealthyIMU: Stealing Permission-protected Private Information From Smartphone Voice Assistant Using Zero-Permission Sensors

Ke Sun, Chunyu Xia, Songlin Xu, Xinyu Zhang
University of California San Diego
kesun@eng.ucsd.edu, cxia@ucsd.edu, soxu@ucsd.edu, xyzhang@ucsd.edu

***Abstract*—Voice User Interfaces (VUIs) are becoming an indispensable module that enables hands-free interaction between human users and smartphones. Unfortunately, recent research revealed a side channel that allows zero-permission motion sensors to eavesdrop on the VUI voices from the co-located smartphone loudspeaker. Nonetheless, these threats are limited to leaking a small set of digits and hot words. In this paper, we propose StealthyIMU, a new threat that uses motion sensors to steal permission-protected private information from the VUIs. We develop a set of efficient models to detect and extract private information, taking advantage of the deterministic structures in the VUI responses. Our experiments show that StealthyIMU can steal private information from 23 types of frequently-used voice commands to acquire contacts, search history, calendar, home address, and even GPS trace with high accuracy. We further propose effective mechanisms to defend against StealthyIMU without noticeably impacting the user experience.**

I. Introduction

Voice User Interfaces (VUIs) allow a user to interact with a smartphone through voice/speech commands, which significantly improves the smartphone's usability and accessibility. Voice Assistants (VAs), *e.g.*, Google Assistant and Apple Siri, and voice-guided navigation apps, *e.g.*, Google Maps and Apple Maps, represent the most popular apps that employ the VUIs. To function properly, these apps have to access many strong permissions related to user privacy, *e.g.*, calendar, contacts, locations, microphone, SMS, and storage. Then they can vocally respond to user queries through the built-in loudspeaker. Such VUI responses often contain sensitive private information (see examples in Table I).

On the other hand, motion sensors, *e.g.*, accelerometer and gyroscope, are co-located with the loudspeaker, and can potentially create a side channel to eavesdrop on the loudspeaker through vibration sensing [1]–[5]. These sensors pose an alarming threat especially since they are accessible by any app on mainstream mobile OS (*e.g.*, Android and iOS) without user permission. However, existing threats mainly target on classifying a small set of digits and hot words [1], [2], rather than natural speech, from the side channel. The potential risk remains unclear in real-world scenarios.

Network and Distributed System Security (NDSS) Symposium 2023
27 February - 3 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.24077
www.ndss-symposium.org

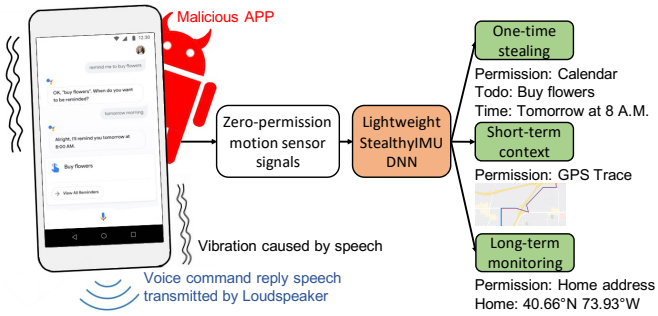


Figure 1. StealthyIMU threat model. StealthyIMU is a malware that can be built in as an ordinary app to continuously capture the motion sensor signals, from which it extracts permission-protected private information.

In this paper, we propose StealthyIMU, a novel and practical threat that *uses the zero-permission motion sensors to extract permission-protected private information from the VUI responses.* These permissions are explicitly granted to the VUI by the user and are strongly associated with user privacy. StealthyIMU is a malware that can be built in or disguise as an ordinary app, to indirectly acquire such permissions through the motion sensor side channel. Fig. 1 illustrates our basic threat model. StealthyIMU continuously captures the motion sensor signals (MSS) and identifies the segments associated with the VUI response, from which it recognizes the type and content of the private information. Over time, StealthyIMU can accumulate more context to further improve the accuracy and richness of the information.

To realize StealthyIMU, we need to resolve 4 key challenges. *(i) How to single out the MSS segments of interest with negligible overhead?* First of all, since StealthyIMU does not know when the VUI responses occur, we need to identify the MSS segments associated with VUI responses, out of all the motion signals, without being noticed by the user. Processing the MSS in the cloud is not always feasible since the StealthyIMU app may not have "network in the background" permission to continuously upload the sensor data. We thus design an on-device two-stage (temporal and frequency domain) detection algorithm, which can identify and segment the sound-induced MSS in the background, and then save them in the app memory with negligible overhead. We further design a lightweight DNN model to single out the subsets of segments that contain an actual VUI response.

(ii) How to recognize the private content from a segment of MSS containing a VUI response? Due to the limited sampling rate of smartphone motion sensors (< 500 Hz), recognizing general speech from the motion sensor is a highly underdetermined problem [1], [2]. Unlike existing work, our unique observations are: 1) VUI responses usually come from a small set of machine-rendered voices, resulting in a constrained *user dependent* speech recognition problem; 2) the acoustic characteristics and format of VUI responses are *more deterministic* than natural human speech, making it easier to be recognized. Since the attacker cares more about the meanings rather than wordings, we formulate the StealthyIMU attack as an end-to-end Spoken Language Understanding (SLU) problem, which further evades the need for word level recognition. Specifically, we extract the contents associated with private permissions from the VUI responses and label them in the form of *private entity list* (see Tab. I). Then we design a DNN model to recognize the private entity lists from MSS. To enhance the SLU, we propose a *cross-modality teacher-student training* strategy to distill knowledge from data-rich speech models to guide the training of the MSS model.

(iii) How to extract the private information by combining the contextual information across multiple VUI responses? The one-shot SLU from a single VUI response is inevitably error-prone. However, by processing multiple VUI responses targeting the same permission, we can infer the user privacy (*e.g.*, city name and home address) with high accuracy. In addition, by inferring the user's city and combining the city map along with the contextual information from a series of navigation voices, we can even recover the user's GPS trace.

(iv) How to defend against the StealthyIMU attack? The limited sampling rate of motion sensors brings opportunities to counteract the StealthyIMU attack by modifying the VUI response speech alone. We propose to *pre-distort* the low-frequency components of speech signals to prevent the motion sensor from capturing the voice-related information, without noticeably affecting the speech quality of the VUI response. Specifically, we design a speech signal processing pipeline to reduce the SNR of the MSS and distort the spectrogram by adding artificial low-frequency components.

To evaluate StealthyIMU, we develop an Android app to collect more than $45,000$ VUI responses from Google Assistant and Google Map, along with the MSS in the real world. The VUI responses include 23 types of frequently-used voice commands (see Tab. II). Our evaluation shows that, for the one-shot attack, StealthyIMU can extract the private entities including contacts, location, reminder/ alarm TODO list and time, and search history with an average 85.55% success rate. For long-term monitoring, StealthyIMU achieves 99.8% success rate in recognizing a user's city name via 5 weather-related queries and locates the user's home address with 11 m average error simply through 10 navigations to home. For short-term contextual inference, StealthyIMU can combine the city road map and a series of navigation voices to recover users' GPS trace within 30 m distance error in 80% of cases. Whereas StealthyIMU can be directly deployed on smartphones, we also take the first step to exploit alternative deployment models of StealthyIMU (*e.g.*, in the cloud), and show the trade-off between the required permissions and on-device system resources. On the other hand, our defense approach is able to prevent the StealthyIMU attack, reducing its SNR down to 2.7 dB and success rate to less than 4.94%. For the purpose of reproducing our approach, we release our labeled VUI response, MSS dataset, and the source code [1].

We make the following contributions in this work:

- We introduce a new threat model that uses zero-permission motion sensors to extract permission-protected private information from VUI responses on smartphones.
- We formulate the StealthyIMU attack as an SLU problem and design a sequence-to-sequence DNN model with a cross-modality knowledge distillation strategy to directly extract the private entities from the MSS. Our model is optimized for on-device execution with minimal overhead.
- We design algorithms to realize the short-term and long-term StealthyIMU attack, and demonstrate how it steals user calendar, GPS trace, home address, *etc*. with extensive experiments in real-world scenarios.
- We propose a speech pre-distortion mechanism to defend against StealthyIMU without noticeably affecting the VUI speech quality.

We note that, even if future smartphone OS restricts the motion sensor permission, the StealthyIMU attack can still work—it can pretend to be an innocuous app that needs the motion sensor permission alone, while using it to extract other permission-protected sensitive information.

II. Related Works

A. Motion Leakage via Sensors on Smartphone

To facilitate common user interaction functions (*e.g.*, screen auto-rotation and app layout rendering), motion sensors are designed to be accessible by arbitrary smartphone apps without explicit permissions. Prior work has investigated various security/privacy threats associated with such zero-permission sensors. ACCessory [6] and Adam *et al.* [7] use the accelerometer as a side channel to extract text input, PIN, and unlock pattern on a smartphone touchscreen keyboard. AccelPrint [3] and TapPrints [8] show that accelerometers and users' tapping both possess unique fingerprints and can be used to identify the users. Mole [9], Liu *et al.* [10], Wang *et al.* [11] and Snoopy [12] investigate the motion leakage via smartwatch motion sensors. Further, motion sensors can be used to infer the user's moving trajectories [13]–[16]. ACComplice [13] records the motion sensor signals for more than 15 mins to recover a relatively long trajectory, and then fits the trajectory to a map based on the shape of the trajectory. Narain *et al.* [14] further designed a graph theoretic model to infer the users' trajectory via accelerometers with 30% top-10 route accuracy. Hua *et al.* [15] demonstrate that accelerometer data can indicate train location. PinMe [16] combines sensory and non-sensory data to infer user location. PowerSpy [17] proposes to use the power consumption change caused by the phone cellular modems to track the user location. Although these attacks are related to the location permission attack in StealthyIMU, they assume either the attacker knows the user's initial location, or the victim is traveling along a small set of known routes. In comparison, StealthyIMU can extract

[1] https://github.com/Samsonsjarkal/StealthyIMU

the location by using a single navigation voice, recover the user's GPS route by combining multiple navigation voices, and even reveal sensitive locations such as home addresses without knowing the user's initial location. Further, it can be combined with relative motion tracking [14]–[16] to infer more precise user locations. Besides, StealthyIMU can steal a much wider range of private information than motion leakage attacks.

B. Speech Recognition via Motion Sensors

Typically, the voiced speech of an adult male and female has a fundamental frequency $85 \sim 180$ Hz and $165 \sim 255$ Hz, respectively. A small portion of the low-frequency speech signals can be captured by the smartphone motion sensor (typical sampling rate $200 \sim 500$ Hz). Prior research exploited this side channel to recognize speech from the loudspeaker vibration. Gyrophone [18] achieves about 50% success rate in identifying 10 speakers, and 65% and 26% for speaker-dependent and speaker-independent 10-digit speech classification. Accelword [4] achieves a hot word detection accuracy of 85% in static scenarios and 80% in mobile scenarios among 10 users. Their threat model assumes that the loudspeaker and the attacking sensor are separated but share a common surface (*e.g.*, desktop) which is not always feasible. Speechless [19] analyzes the MSS side channel and shows that through-air human speech does not noticeably affect the motion sensors. Recently, Spearphone [2], [20] and AccelEve [1] further examined the feasibility of using smartphone motion sensors to recognize the speech reverberations from a co-located loudspeaker. StealthyIMU differs from existing work in two aspects. First, the threat model and end goal are different. Previous work only demonstrates the possibility of using MSS to recognize a predefined set of numbers or words. In contrast, StealthyIMU can extract complete semantic information. It reveals a real-world privacy threat where MSS can steal crucial smartphone permission-protected private information, *e.g.*, GPS trace, location permission, calendar, contacts, *etc*. Second, the attacking vector and methods are different. Previous work cast the eavesdropping of numbers/words into a simplified classification problem. In comparison, we formulate the StealthyIMU attack as an end-to-end private speech understanding problem, and design speech and natural language processing algorithms to efficiently extract the private information from the MSS side channel. We also investigate the one-time stealing, short-term contextual inference, and long-term monitoring threat models.

C. Spoken Language Understanding

SLU systems infer the intents and meanings of a spoken utterance [21]. This is critical for VUIs, which convert the speaker's utterance into action or query. SLU typically consists of two tasks: Intent detection, a classification problem where utterances are labeled with predefined intents; Slot filling, a sequence labeling task that identifies the semantic concepts. The basic problem behind StealthyIMU is close to SLU. It tries to first classify the leaky permission (corresponding to intent detection) and then recognize the private information (corresponding to slot filling) from the MSS. Existing SLU systems can be categorized into two classes: a cascaded pipeline approach and an end-to-end approach [22]. The former contains an automatic speech recognition (ASR) system to decode the speech into text, followed by a natural language understanding (NLU) system that interprets the meaning of the text [23]. The performance highly depends on the accuracy of the ASR stage, whose error may be propagated and amplified in the NLU [24]. The latter approach uses a single DNN model to map the speech signals directly to the speaker's intent without an explicit text transcription [22]. In this paper, we systematically analyze the unique challenges of StealthyIMU, and choose to design an end-to-end model to extract the private permission from the MSS (Sec. V-A).

III. Threat Analysis

To function properly, VUI apps have to access many strong permissions related to user privacy. For example, Google Assistant requires 7 permissions, including calendar, contacts, locations, microphone, phone, SMS, and storage. StealthyIMU targets the scenario where a malware uses zero-permission motion sensors to record the vibrations caused by the voice assistant and voice-guided navigation apps, and then steals the permission-protected private information from these apps. To establish the threat model, we assume the adversary can mislead the victim to install an ordinary app that contains the StealthyIMU malware, which then continuously collects MSS in the background. Once the app receives the MSS, it will detect whether there exist any VUI responses, recognize the types of responses, infer the private entities from the responses, and recover the explicit results of the stolen permission-protected private information. Finally, these information will be saved in the app locally or uploaded to a cloud server (if permitted), to monitor the user's real-time location and trajectory, understand the user's daily schedule (*e.g.*, medication input), personal habits and preferences, *etc*. Notably, popular mobile browsers such as Firefox allow motion sensor access by default. So, besides hiding itself in a normal app, StealthyIMU can also be launched through an executable (*e.g.*, Javascript) on a seemingly innocuous website.

Attacking requirements: The StealthyIMU attacker needs no explicit permission to capture the MSS. However, it needs to ensure neither the smartphone OS nor the user will notice the attack. To this end, it needs to carefully make several trade-offs related to system resource requirements.

• *Voice detection on device v.s. continuous network streaming*: Since StealthyIMU does not assume to know when the motions come from the VUI, the first step of StealthyIMU is to single out the VUI-induced motion signals from all the MSS. Although such computation overhead of StealthyIMU can be completely outsourced to the adversaries' cloud server, it is not reasonable to assume that the malicious app always has the "network in the background" permission to continuously upload the MSS stream. Consequently, the malicious app should be able to perform on-device identification of the MSS associated with VUI responses, with minimal overhead. Note that StealthyIMU still needs "network" permission to upload private information to adversaries. However, the on-device processing helps StealthyIMU to avoid continuous network streaming. Uploading private information to adversaries can be accomplished when the malicious app is active, and the uploading data size is significantly reduced compared to uploading the MSS stream.

• *Memory v.s. storage*: After identifying the MSS of interest, StealthyIMU can choose to save the segment of signals

Voice Command		Permission	Examples	Private Entity List
Set/Check calendar/reminder		Read calendar	Q: Set a reminder to check bank account. A1: Got it. "[Check bank account]". When do you want to be reminded? Q: Tomorrow 8 PM A2: Sure. I will remind you [tomorrow at 8 PM].	A1: Calendars [Todo: Check my bank account] A2: Calendars [Time: 8 PM]
Make calls and send messages		Read contacts Read SMS	Q: Send a message to my father. A3: Sure. What's the message? Q: Call me back. A4: Sending a message to [Bob] saying "[Call me back]".	A3: A4: Message [Contacts: Bob, Content: Call me back]
Ask Question	Weather	Access coarse location	Q: What's today's temperatures? A5: Temperature in [New York] tonight is predicted to be 54 °F.	A5: Weather [City: New York]
	Navigation		Q: Navigate me to Los Angeles. A6: OK. [Los Angeles]. Let's go.	A6: Navigation [City: Los Angeles]
	Stock, Sports, Music	Access search history	Q: What is the stock price? A7: [Amazon.com] at XXX dolloars.	A7: Search [Company: Amazon]
Navigation APP		Access fine location	A8: In 600 feet use the right lane to [turn right] onto [Hollywood Boulevard]. A9: Use the left two lanes to [turn left] onto [Western Avenue].	A8: Navigation [Intent: Turn right, Road: Hollywood Boulevard] A9: Navigation [Intent: Turn left, Road: Western Avenue]

Table I. StealthyIMU potential attacking permissions and VUI response examples. StealthyIMU proposes to use the MSS caused by the responses of the VA (In the "Examples" column, utterances start with "A") to infer the permission-protected private information from the VUI.

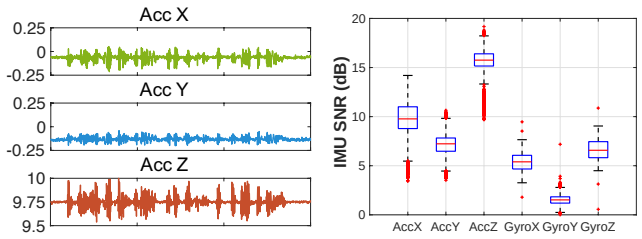
in the app memory or local storage, depending on whether it has the "storage" permission. Note that if the app saves the segment in memory, only a small amount of memory is needed (16 KB per VUI response on average. see Sec. IX-F), so it is indistinguishable from an innocuous app.

This paper takes the first step to exploit these choices and navigate the trade-off between required permissions and on-device resources. *We design StealthyIMU such that it can steal private information even without continuous network streaming and "storage" permissions*, by only using on-device computation. On the other hand, if StealthyIMU has access to the continuous network streaming and "storage" permissions, it can execute the majority of the attack vector on the cloud.

Target permissions: We investigate frequently used voice commands related to reading the VAs' permissions [25] and summarize a list of vulnerable permissions and example attacks in Table I. We put these privacy-related voice commands into *3 categories*, *i.e.*, set/check calendars and alarms, make calls and send messages, and ask questions. In addition, we put voice-guided navigation in a separate category, as it only utters the navigation voices without a conversation with the user. StealthyIMU can steal 5 reading permissions, *i.e.*, reading the calendar, contacts, SMS; accessing the location (coarse and fine), and search history. The "Examples" column shows the conversational examples between the user (Utterances starting "Q") and the VA (starting with "A"). The last column, "*Private Entity List*", shows the information that StealthyIMU aims to extract from the MSS. With the inferred private entities, StealthyIMU indirectly steals the permission-protected private information from the VA and the voice-guided navigation app.

IV. Motion Sensor Signal (MSS) Preprocessing

In this section, we introduce the lightweight on-device preprocessing mechanisms to prepare StealthyIMU for an attack. Basically, we first select the motion sensor channels based on the SNR. Then we design a near-zero overhead algorithm to continuously detect and segment the sound-induced samples in the MSS. Finally, we use a voice identification model to identify whether a detected segment is associated with a VUI response from the co-located loudspeaker.



(a) Acc waveform (b) Channel SNR

Figure 2. Motion sensor channels SNR

A. Choosing the Sensor Type and Channel

Figure 2 shows the accelerometer signal waveform when a smartphone (*i.e.* Samsung S8) loudspeaker is playing the VUI response at 80% volume. We quantify the motion sensor's response to sound by $\text{SNR}_{dB} = 10\log_{10}\frac{P(S)}{P(N)}$, where $P(\cdot)$ is the sum of squared magnitude values; S and N are two series of MSS recorded with and without the loudspeaker voice. Prior work [1] showed that the speech-induced vibration may cause different SNR on different sensors and sampling channels. The accelerometer's Z axis signals always have the strongest SNR, regardless of the smartphone placement and sound volume, because the vibration generated by the loudspeaker is always perpendicular to the smartphone's motherboard (along Z axis) [1]. Although using all the IMU sensors for StealthyIMU may improve its performance, it will also increase the attacking overhead significantly. *We thus choose the accelerometer's Z axis signals alone as the input in StealthyIMU.*

Existing IMU-based sensing systems [1], [20] save an entire stream of signals locally and preprocess it offline. However, as discussed in Sec. III, StealthyIMU may not always have permission to do so. Instead, we propose a real-time signal processing workflow to detect and segment voice-induced accelerometer signals with minimal computation and memory overhead. Meanwhile, our detection algorithm needs to achieve a high true positive rate, while maintaining a reasonable false positive rate in order to limit memory usage.

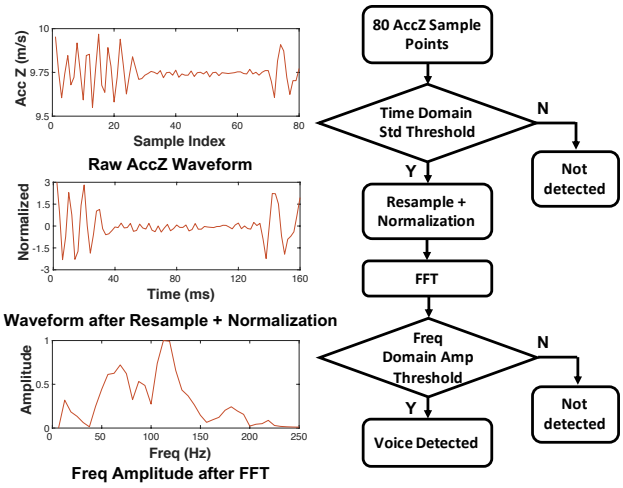


Figure 3. Ultra Lightweight Voice Detection Pipeline

B. Real-time Detection and Segmentation of Voice in MSS

One major challenge for preprocessing lies in the unstable sampling frequency of the motion sensors, which varies over time and across different smartphones. Prior work [1] resamples the accelerometer signals and normalizes to the same sampling frequency frame by frame by using linear interpolation, which incurs too much computational overhead (1 multiplication and 2 additions per sample). In contrast, our voice detection is an ultra lightweight two-stage algorithm. The first stage detects the vibration of interest based on an empirical *std.* threshold, to rule out the cases where the smartphone is either static or moving abruptly. The second stage resamples the signals to 500 Hz, and then normalizes the magnitude values. Resampling here incurs an acceptable computational overhead since only a small fraction of MSS passes the first stage. Then, we apply a Discrete Fourier transform (DFT), and remove those segments that do not contain significant high-frequency components.

Finally, we need to segment the signal buffer associated with an entire voice sequence. Based on a real-world large scale data set that we collected (Sec. VIII-A), we found most VUI responses last 2 s to 8 s. Thus, we choose to keep a signal buffer of 4000 points (8 s duration at 500 Hz) in the app memory. Within this buffer, we count the number of 80-point segments that are detected as voice-associated. If the detection is positive in the first 80-point segment of the buffer and there exist more than k such segments ($k = 2$ as default), the buffer will be considered as potentially containing the VUI response and stored in memory. Note that a single buffer of float type samples only consumes 16 KB (4000×4 Bytes), which means we can save 500 potential segmented buffers with a small 8 MB memory.

C. Feature Extraction

The feature extraction module first resamples the signal segments to 500 Hz, and then performs an STFT to obtain the spectrogram. The STFT uses a window length of 80 ms and a hop length of 20 ms to guarantee 50 frames per second. This is larger than typical speech recognition window size (25 ms) and hop length (10 ms) because the sampling rate of MSS is much lower. Finally, to mitigate the accelerometer signal noise due to motion artifacts, we remove the frequencies lower than 62.5 Hz following [26], [27], and create 30 frequency bins within the frequency range of $(62.5, 250]$ Hz. The resulting features are 50×30 scalars per second, which serve as input to the DNN model in Sec. IV-D and Sec. V-B.

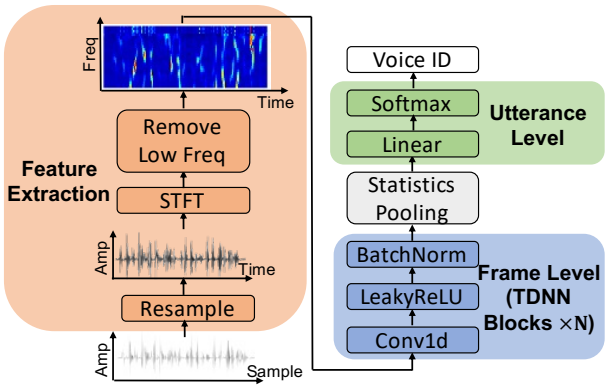


Figure 4. Feature Extraction and Speaker Identification

D. VUI Response Identification

A segment of sound-induced MSS does not always contain a VUI response. It could originate from other sources such as phone calls. To single out the segments of interest, we observe that there are only a limited set of voice profiles available for VUIs, *e.g.*, 5 male and 5 female voices on Google Assistant. Thus, we can simply identify if the MSS contain a VUI response from a specific voice ID. We design a lightweight DNN model (Fig. 4), inspired by XVector [28], to address this problem.

Specifically, the model takes the aforementioned spectrogrm feature as input. The output contains 11 classes, including 10 different voices from Google Assistant and a class for other sound sources. We first embed the frame level feature by using TDNN blocks [29], and then apply an attentive statistics pooling layer [30] to convert the variable length frame-level features into a fixed-dimensional vector, referred to as deep speaker embedding in text-independent and variable-duration scenarios [31]. Finally, the deep speaker embedding is fed into an utterance-level feature extractor comprised of fully-connected hidden layers.

To train the VUI response identification model, we use an accelerometer data set collected on COTS smartphones, which consists of 10 hours of VUI responses for each voice profile, and 35 hours of accelerometer signals when playing Youtube Videos. Our benchmark experiments (Sec. IX-A) show that StealthyIMU achieves 3.41% EER for identifying the targeted VUI voice by using a lightweight model of less than 2 MB. Our experiments show that StealthyIMU achieves 3.41% EER for identifying the targeted VUI voice by using a lightweight model of less than 2 MB.

V. Inferring Privacy from a Single VUI Response

In this section, we discuss the problem formulation, DNN model design and training strategy to infer the private entity list from a singe VUI response.

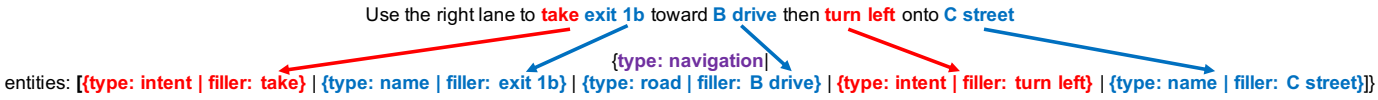


Figure 5. Example to convert the navigation voice text to private intents

A. Problem Statement

Recognizing general speech from the MSS is a highly underdetermined problem, as the sensor only samples the vibration artifacts and at much lower sampling rate than microphones. However, two key observations enable StealthyIMU to circumvent the barrier. First, there are only a limited set of machine-rendered voice profiles and the user rarely switches between them. So the VUI responses almost always come from the same voice, making the speech recognition problem *user-dependent*. Second, the acoustic characteristics of VUI are more deterministic than natural human speech.

Inspired by recent research in speech and natural language processing, we propose to formulate the StealthyIMU attack as a Spoken Language Understanding (SLU) problem. SLU systems are widely used in VUIs which recognize the intents and meanings of speech directly, rather than word-by-word transcription. In comparison, StealthyIMU takes the MSS as input, and extracts a private entity list within the VUI response, as exemplified in Table I. Since word-level recognition using MSS is error prone [19], StealthyIMU designs an end-to-end SLU DNN model to directly interpret the VUI responses.

DNN input, output, and ground-truth generation: The input into our SLU model is the feature map that has been obtained after the voice response identification and feature extraction (Sec. IV-C and Sec. IV-D). The output is the text of private entity list, which comprises *i)* voice response type and *ii)* private entities. The machine-rendered VUI responses tend to follow a fixed format. The same type of responses often contain the same number and pattern of private entities, as shown in Table I. Navigation voice is the most complicated type, where the sentence structure is consistent but the entity number varies. We thus take the navigation voice from Google Map as an example to explain the general steps of how we generate the *ground-truth* private intent lists.

First, by empirically analyzing the grammar of the real-world navigation voices in our data set, we summarize the private entities of interest as follows:

i) Intents, *i.e.*, the next driving intent. There are mainly 16 intents, *i.e.*, "turn left", "turn right", "take the next left", "take the next right", "slight left", "slight right", "keep left", "keep right", "continue", "stay", "take" (highway and exit), "make a u-turn", "merge", "follow sign", and "arrive at destination".

ii) Name, *i.e.*, road, highway and exit names.

Second, we take the grammar into consideration. Fig. 5 shows an example of converting the navigation text to private intent. We first add the voice response type, *i.e.*, navigation. Then, we identify the private entities including both intent and name. All of the entities except voice response type are in the text and are added into the private intents in natural spoken order. This workflow of generating the ground-truth also gives the model prior knowledge about the sentence structure and potential entities' locations within the sentence.

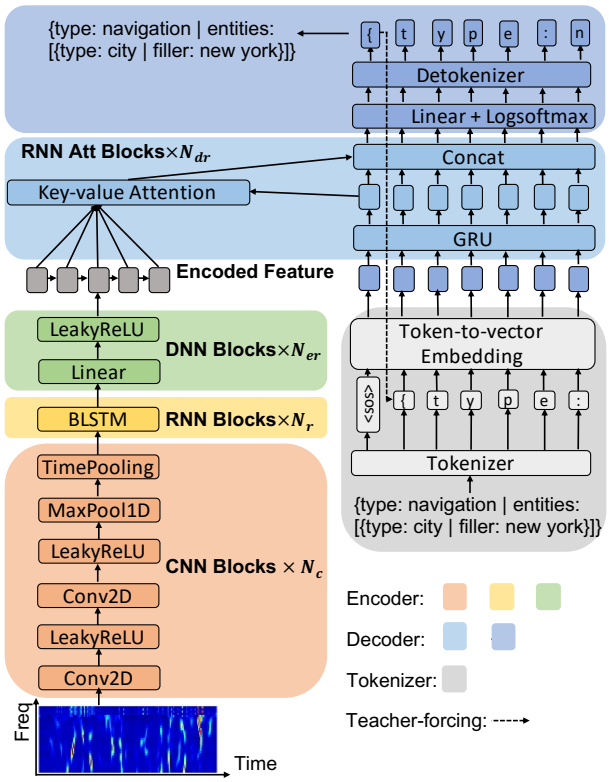


Figure 6. Model Architecture

B. Model Design

Our end-to-end SLU DNN model is a sequence-to-sequence model, which comprises 3 components, *i.e.*, Tokenizer, Encoder, and Decoder. As shown in Fig. 6, the input of the model is represented as a sequence of the frequency spectrum, $\mathbf{x} = \{x_1, x_2, .., x_t\}$, where t is the temporal length of the spectrogram of the voice-associated MSS. The output is a sequence of tokens corresponding to the private entity list, $\mathbf{y} = \{y_1, y_2, .., y_s\}$, where s determines the length of the list.

Tokenizer: To optimize our model towards the ground truth, we need to quantitatively encode the text of the private entity list. We use a tokenizer to break the raw private entity list into tokens, which can be word-level, subword-level, or character-level. Since the lexicon of the private entity is large (> 4000), using word-level token will increase model complexity. Further, the relationship between the time-frequency (T-F) spectrogram and the word-level tokens is hard to learn for a DNN model. In comparison, subword-level and character-level tokens are closer to the *phoneme*, the basic unit of human voice, which has proven to be suitable for speech processing [32]. To reduce the model size and improve the recognition rate, we choose to create character-level tokens using the Google SentencePiece Tokenizer [33]. The corresponding detokenizer will be used in the inference stage to recover the private entity list from the token sequence.

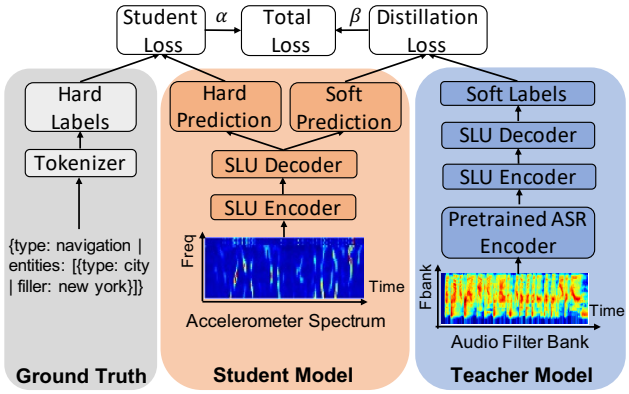


Figure 7. Knowledge Distillation Training Strategy. StealthyIMU proposes to distills the knowledge from the speech model to guide the training of the motion sensor model. The "Student Model" corresponds to the model trained by MSS as shown in Fig. 5. The "Teacher Model" has the similar architecture with "Student Model", but is trained by speech signals.

Encoder: The input sequence $\boldsymbol{x}$ is fed into the encoder to embed the T-F spectrogram feature. Fig. 6 shows the structure of our encoder, which contains N_c CNN blocks, N_r RNN blocks, and N_{er} DNN blocks. We use the "TimePooling" layer in each CNN block to reduce the length of the input sequence by half. Except for the TimePooling layer, the encoder does not change the length of the input sequence, resulting in hidden state length $t/(2^{N_c})$.

Decoder: Our decoder model contains N_{dr} RNN attention blocks (Fig. 6). The embedded feature of the last token is first fed into a GRU layer. To leverage the entire encoded hidden state from the encoder, we use a key-value self-attention mechanism [34], which takes the encoder hidden state as the keys and values and the decoder hidden state as the queries for the self-attention layer. Finally, the hidden state with attention is concatenated with the original decoder hidden state as the output of the RNN attention block along with the linear and softmax layer to recognize the output token sequence $\hat{\mathbf{y}}$.

C. Training Strategy

Teacher-forcing: We apply the teacher-forcing training strategy to train the decoder, which uses the ground truth token from a prior output token as input to the decoder, to prevent error accumulation. We adopt the negative log likelihood loss function $l_s = \max_\theta \sum_i \log P(y_i|\mathbf{x}, y^*_{<i}; \theta)$, where $y^*_{<i}$ is the ground truth of the previous tokens.

In the inference process, we replace the teacher-forcing by using the last predicted token as the input of the decoder. We use beam search [35] with beamwidth 40 to generate the sequences of tokens with the highest probability globally.

Knowledge Distillation from Speech: To enhance the SLU, we design a teacher-student cross-modal knowledge distillation DNN, which distills the knowledge from the speech model to guide the training of the motion sensor model. The rationale behind this design is two-fold. First, the speech model is more feature rich and hence accurate than the motion sensor model. Thus it can facilitate the motion sensor model to learn reasonable attention across an entire segment of MSS. Second, due to the massive amount of training data, the speech model is more general than the motion sensor model, which helps overcome the lack of training data for the MSS.

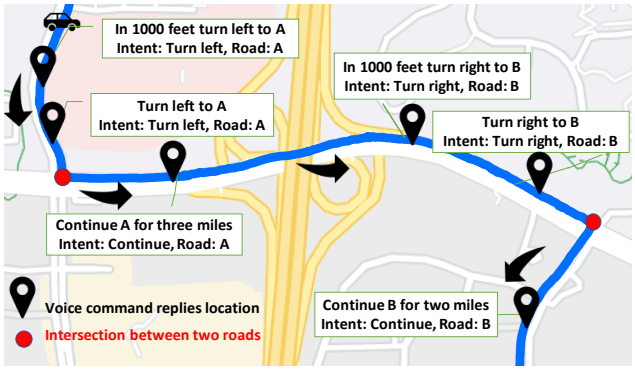


Figure 8. Trace and commands in navigation

As shown in Fig. 7, we simultaneously collect the speech and accelerometer signals associated with the voice responses, through the microphone and accelerometer, respectively. We first train the teacher SLU model by using the speech signals as input. Specifically, we use a *pretrained* ASR encoder, trained by the LibriSpeech dataset [36], as a basic encoder. Then, an SLU encoder and decoder, similar to the model in Sec. V-B, are used to train this speech-based teacher model.

The student SLU model uses the corresponding accelerometer signals as input and has been introduced in (Sec. V-B). During the training of the student model, we freeze the teacher model layers and achieve cross-modal knowledge distillation by narrowing the distance between the output distribution of the two models. We choose to distill the knowledge from the decoder output because the outputs of the corresponding speech signals and accelerometer signals are expected to be the same. We use the Kullback–Leibler (KL) divergence loss [37] as our knowledge distillation loss function $l_d = \frac{1}{s}\sum_{i=1}^{s} KL(P_S^i, P_T^i)$, where s is the output sequence length, P_S^i and P_T^i are the output distribution of the student model and teacher model respectively. Finally, the combined loss function is: $l_{total} = \alpha l_s + (1-\alpha) l_d$. We set $\alpha = 1$ in the first few epochs, and then gradually reduce α to increase the weight of knowledge distillation loss and help convergence.

We note that the teacher-student training strategy is only used in the offline training process, and does not increase the model size and computation overhead.

VI. Extracting Permission-protected Privacy

In this section, we explain how to extract permission-protected private information by extracting and combining the private intents from single or multiple VUI responses. Based on the availability of contextual information, we categorize the attack model into three different scenarios.

A. One-Time Stealing

One-time stealing means that StealthyIMU only takes a single VUI response as the input to extract privacy information. Voice commands (*e.g.*, set/check calendars and alarms, make calls and send messages, and search for stock, sports, and music) are usually resolved in a single response, without any contextual information. The private intents inferred from the

response directly correspond to the private permissions. Such one-time stealing achieves a relatively lower success rate than the other two scenarios since it fully depends on the SLU from a single voice response. In Sec. IX-B, we demonstrate that StealthyIMU achieves an average 85.28% success rate across 12 types of voice commands and is also able to identify other 11 types of voice commands without explicit private information.

B. Short-Term Contextual Inference

Short-term contextual inference represents the scenario where StealthyIMU can infer the private information by using multiple consecutive ($\geqslant 2$) VUI responses. One of the most frequently-used cases is the navigation app. To direct the user towards the destination, the navigation app uses a sequence of navigation voices during navigation. Typically, for each turning point, there exists 1 or few navigation voices, which provide contextual information continuously, allowing StealthyIMU to track the user's location and even recover the GPS trace. In this section, we show how StealthyIMU extracts a sequence of private intents from a navigation process, combined with the city-scale road map from OpenStreetMap [38], to recover the GPS trace. Note that 2 consecutive VUI responses are sufficient to infer the GPS location of the victim. To recover one GPS route trace, the duration of eavesdropping depends on the navigation time of this route, which ranges from a few minutes to a few hours with $2 \sim > 50$ VUI responses.

Fig. 8 shows an example. Each "black" mark represents a navigation voice that happens along the route. StealthyIMU can already extract both the "intent" and "road name" of each navigation voice following the workflows in Sec. V. The key problem here is to recover the "blue" trace. To this end, we design a GPS trace recovery algorithm (see Algorithm 1). Our basic idea is to first identify the intersection between two roads (red dots in Fig. 8) by checking the ordered list of road names that we extracted. Then, we can directly connect each two consecutive intersection points by using the shortest path algorithm on the road map.

To enable private information extraction, the GPS trace recovery algorithm needs to address two challenges uniquely related to StealthyIMU. First, the intent recognition is not perfect while the navigation app itself may also produce occasional incorrect navigation voices due to poor GPS signals. Therefore, sometimes we cannot find the intersection between two consecutive roads because of a missing or wrongly recognized road. To improve robustness, we design a *correction mechanism* to interpolate/skip a road name to prevent the false recognition and enable the algorithm to find as many intersections as possible (Line $3 \sim 10$). Further, we observe that for each "turning" intent, the navigation app repeats the navigation voice twice, before the turn and right at the turn. We leverage this redundancy to double-check the road name.

Second, our method can not recover the source or destination points for the route due to a lack of corresponding intersections. To overcome this problem, we utilize the intent information to find the names of the first and last road segments. Then, we use the time interval between the first/last navigation voice and the first/last turn navigation voice to estimate the starting point and destination (Line $14 \sim 21$). Besides, the intents of each navigation voice can also help check whether the shortest path follows the correct intent. If they are different, we will add another point after the turn intent to the intersection list to correct the GPS route (Line $22 \sim 25$).

Algorithm 1: Trace Recovery Algorithm

```
   Input: N commands stolen in whole navigation
   Output: Complete trace
 1 Extract (intent, road) pairs that appear at least twice in succession;
 2 Preprocess the pairs by cleaning, integrating, and sorting;
 3 foreach road in road list do
 4     if two disjoint roads have shared road then
 5         Interpolate the shared road to the road list;
 6     end
 7     if two roads intersect by skipping other roads in between then
 8         Remove the skipped roads in the road list;
 9     end
10 end
11 Find all intersection points for the roads in order;
12 Connect the point sequence to recover the trace with the shortest
     path;
13 foreach (intent,road,point) in trace do
14     if first intersection point then
15         Infer the direction of the initial path with the intent;
16         Estimate the starting point based on the time interval.
17     end
18     if last intersection point then
19         Infer the direction of the final destination with the intent;
20         Estimate the destination based on the time interval.
21     end
22     Check the direction in the shortest path with the intent;
23     if direction in the shortest path is against the intent then
24         Take the opposite direction of the same road;
25     end
26 end
```

C. Long-Term Monitoring

Long-term Monitoring represents the scenario where the users repeat the same type of voice commands in a few days, like check weather, air quality and reminder, etc., while StealthyIMU runs the intent extraction workflow continuously to steal privacy. Although Sec. VI-A has shown that StealthyIMU can extract the private information with a single VUI response, the long-term monitoring aims to accumulate its knowledge over time and improve the accuracy of privacy extraction. Specifically, we first use the SLU model to detect the voice command type (Sec. V-B). Then, we calculate the probabilities of the top-n private entity list output. Since each voice response is a single individual event, the final success rate of an attack is the probability of multiple independent events in the long run.

For a more clear exposition, we take *city name extraction* as an example. When asked about the weather, the VUI will report the weather along with the city name. For a single VUI response, StealthyIMU takes 120 US cities with more than 200, 000 population each as the potential city list. Then, for each potential city name, StealthyIMU takes the specific MSS of a single VUI response y_0 as the input and uses the DNN model in Sec. V to calculate the probability $P(x|y_0)$ of the output token sequence of that city, where x is the token sequence of that potential city. With multiple such queries over a few days, StealthyIMU calculates the probability of each potential city as $P(x|y) = 1 - \prod_1^K (1 - P(x|y_i))$, where x is still the token sequence of that potential city, K

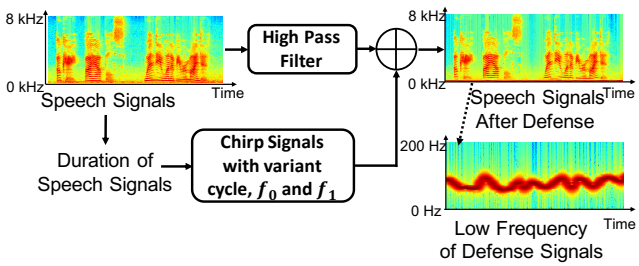


Figure 9. Pipeline of the speech predistortion defense.

is the number of VUI responses; y_i and y denote the MSS signals of i-th VUI response and the set of K MSS signals respectively. StealthyIMU finally estimates the victim's city name by selecting the result with the highest probability to further improve the attacking success rate of StealthyIMU.

Another example is the *home address extraction*. Google Map always utters "Welcome home" when reaching the user's home as the destination. As discussed in Sec. V, StealthyIMU first extract the "road name" and "intent" from each navigation voice. Once StealthyIMU identifies that the "intent" for a specific navigation voice is "Welcome home", StealthyIMU runs the GPS trace recovery (Sec. VI-B) for the previous consecutive VUI responses to locate the destination of the GPS trace. Then, with multiple navigation attempts that contain such an intent to be back "home", StealthyIMU can calculate multiple potential home addresses. Finally, to improve the accuracy of home address extraction, we combine these potential home addresses by first removing the outliers and then calculating the centroid of the remaining GPS points. In Sec. IX-D, we evaluate these examples in the wild to show the threat of long-term monitoring.

VII. Defense Against StealthyIMU

In this section, we propose two defense mechanisms that can thwart the StealthyIMU attack.

A. Predistortion of Speech Signals

Our first defense protects existing smartphones, whose highest MSS sampling rate is < 500 Hz [1]. Our key insight is that *modifying the low-frequency components of speech signals will significantly impact the MSS, without noticeably affecting the human perceivable speech quality*.

Figure 9 shows the pipeline of this approach. First, we use a high-pass filter to remove the low frequency components in the speech signals with a cut-off frequency f_c Hz. We empirically choose $f_c = 350$ Hz based on the measurement in Sec. IX-G, to balance the protection of privacy and reduction of speech quality. Second, we distort the T-F spectrogram of the MSS by injecting a structured low-frequency interference signal through the smartphone's audio interface, more specifically a chirp signal with varying cycle and starting/ending frequency. The cycle range of the chirp is set to $100 \sim 200$ ms because we aim to distort the spectrogram of each phoneme which is the basic unit of voice pronunciation and typically lasts less than 200 ms [39]. The advantage of this approach is that it only needs the VUI app vendors, *e.g.*, Google, Amazon, Apple, to add a pre-distortion stage in their voice responses, without any hardware modification.

Type	Example Voice Command	Privacy	#
Weather	What's the weather today?	Location	12,527
Sun set&rise	What's the sunset in Chennai	Location	1,505
AirCheck	AQI for San Francisco	Location	1,601
Clock	What time is it in London	Location	1,595
Reminders	Set a reminder to check my account	Todo	2,950
	Set a reminder for tomorrow morning	Time	2,140
Media Alarms	Set an alarm to go to fedex	Todo	2,630
	Set a music alarm at 8 PM	Time	2,350
Stock Updates	Stock price for Apple	Search	1,318
Calling	Call Sam	Contacts	1,120
Navigation	Navigate to Los Angeles	Location	1,570
Navigation App	/	GPS	7,885
Fun Tricks	What movies are playing?	Others	500
Sports Facts	What's the news about the NFL?	Others	500
News	What's the news about the covid?	Others	500
Calculations	What calculation can you do?	Others	500
Google Search	How tall is the Eiffel Tower?	Others	500
Youtube Music	Play music on Youtube Music	Others	500
Voice Mail	Call voicemail	Others	500
Youtube	Open Youtube	Others	500
Chrome	Open the Google Trends website	Others	500
Youtube TV	Play FS1 on Youtube TV	Others	500
Broadcast	Broadcast a message	Others	500
Overall			*44,691*

Table II. Types of Voice Command and Dataset Scale.

B. Redesigning the Permissions

Although existing smartphone firmware and driver limit the MSS sampling rate below 500 Hz, the achievable sampling rate of the motion sensor hardware itself can be much higher. For example, the InvenSense MPU 6500 motion sensor, used in Samsung S6 and iPhone 6S, supports up to 4000 Hz. If such high sampling capabilities are unleashed in the future, the above defense approach may fail. We thus highly recommend the smartphone OS to redefine the permissions of the motion sensor. Since most of the apps do not need extremely high sampling rates for motion sensors, the smartphone OS can discriminate the permission levels for different sampling rate limits, and report the potential threats to the users when an app requests access to high-rate MSS.

VIII. Dataset and Implementation

A. StealthyIMU Dataset

To evaluate StealthyIMU, we collect a new benchmark dataset that captures the smartphone MSS when the loudspeaker plays voice responses. Meanwhile, we record the corresponding voice commands in order to generate the ground truth labels and facilitate our model training (Sec. V-C). The trained model will be used later in our real-world attack evaluation.

Voice response data collection: We adopt a natural language voice command text dataset [40], and use Google Text-to-Speech to synthesize the corresponding speech along with the wake word preamble "Hey Google". To collect the dataset automatically, we use an additional device, *i.e.*, a Macbook Pro laptop, to playback the voice commands just like a human user. Meanwhile, a victim's smartphone hears and responds to the voice commands through its loudspeaker while the StealthyIMU app captures the accelerometer signals. We use an open-source test automation framework, called Appium [41], to control the laptop and smartphone and streamline the process. During the data collection, the smartphone is set under two different real-world scenarios, *i.e.*, on a table and a car phone mount, respectively.

As shown in Tab. II, we collected 23 types of VUI responses by asking Google Assistant "What can you do?". Among these, 12 types have fixed structures and explicit permission-protected private information while the other 11 types are unstructured with arbitrary contents. We place these voice commands into 5 categories based on the targeted permissions.

• "Access coarse location": The "Weather", "Sunset & Sunrise", "AirCheck" and "Clock" related responses are to steal the private information from "Access coarse location", *i.e*. city name of the location. We collect a large dataset for "Weather" with $12,527$ VUI responses and relatively small datasets for "Sunset & Sunrise", "AirCheck" and "Clock" with $1,595$, $1,601$, and $1,595$ VUI responses respectively from 120 US cities with more than $200,000$ population each [40]. To generate the training data, we first use the Fake GPS app to change the smartphone GPS to the city, and then playback the these types of voice commands.

• "Read calendar": The set/check calendar/reminder/alarm related responses are used to steal the private information from "Read calendar" permission. We collect $5,031$ and $4,980$ different reminder and alarm commands respectively with 300 frequently-used TODO entities and 300 TIME entities. Since we know the corresponding voice command for each response, we can directly extract the TODO entity and TIME entity and generate the ground-truth entity list.

• "Stock Search": The stock update check command responses are used to steal the private information from "Search history" permission. We collect $1,318$ different stock update commands with top 150 companies with the largest capital in NASDAQ.

• "Contacts": The hands-free calling command responses are used to steal the "Contacts" permission. We collect $1,120$ different calling commands with 200 frequently-used name entities.

• "Others": There exists other 11 types voice commands which has unstructured VUI responses with arbitrary contents. StealthyIMU can not easily extract the explicit private information from these VUI responses. Therefore, we label these VUI responses with "Null" labels.

Finally, we label the ground truth entity list of these voice responses by listening to the recordings. Note that the VUI responses are highly diverse, totaling more than 150 hours and containing more than 4000 frequently-used words. The VUI responses are collected over more than one year.

Navigation app voice data collecting: To collect the navigation app voice with ground truth GPS traces, we first use OpenStreetMap to open a city map. We randomly select two points on the map separated by > 6 km, and use Google Map to generate the navigation route. We export the navigation route to the smartphone, and start the Google Map navigation app to follow this route. Then we use "MAPS TO GPX" to convert the Google Map navigation route to a GPX file which contains the GPS point list of this route. The GPX file is imported to a "Mock Location" app to follow this route by using fake GPS. Finally, Google Map is able to navigate the smartphone from the start point to the destination by checking the fake GPS positions. And again, we use Appium to automate this data collecting process.

To annotate the ground truth of the navigation voice, we design an app to listen to each navigation voice and annotate it based on the potential road name and intent from map and navigation trace. We collect this dataset from two cities.

• City A: a typical city with a population of > 1.5 million and area approximately 800 km^2. We collect 300 routes with $5,091$ navigation voices which contain more than $1,800$ km mileage, 419 roads (95% coverage) and 7 highways (100% coverage) and 55 highway (92% coverage) exits.

• City B: a metropolitan city with a population of > 5 million and area of 800 km^2. We collect 150 routes with 2794 navigation voices which contain more than 900 km mileage, 559 roads (37% coverage) and 8 highways (100% coverage) and 42 highway (84% coverage) exits.

B. Implementation

DNN Implementation: We implement the voice response identification DNN model (Figure 4) and voice response SLU model (Figure 6) in PyTorch. For training, we use the NewBob learning rate strategy [42] at a $3e-4$ initial learning rate followed by annealing. The default training batch size is 8, and the number of epochs is 30. The pretrained models are then converted into Just-In-Time (JIT) PyTorch model running on smartphones.

Attack Implementation We implement two versions of StealthyIMU, one running entirely on an Android phone and the other facilitated by a cloud server. Our app can choose to *i)* detect voice on the smartphone or stream the signals continuously to the cloud; *ii)* save the signals in the memory or local storage; *iii)* execute the attack either on device or in cloud. We evaluate the system overhead and permission risks of these different attack surfaces in Sec. IX-F.

IX. Evaluation

The evaluation of StealthyIMU consists of four parts. First, from Sec. IX-A to Sec. IX-D, we use the default large-scale dataset discussed in Sec. VIII-A for training. And then, we test StealthyIMU by collecting specific testing dataset for each attacking mode. Second, in Sec. IX-E, we collect additional testing datasets with different real-world scenarios and evaluate StealthyIMU across different factors to see whether the model can generalize to new scenarios. Finally, we evaluate the system overhead and defense mechanisms of StealthyIMU in Sec. IX-F and Sec. IX-G respectively.

A. DNN Model Ablation Study

We first conduct an ablation study to evaluate the accuracy and efficiency of the StealthyIMU DNN models, *i.e*., VUI response identification (Sec. IV) and VUI response private entity recognition (Sec. V). In this study, we use the dataset containing 23 types of VUI responses, as discussed in Sec. VIII-A. We mixed all of the VUI responses and train a single model to extract the private information from a VUI response across different types of VUI responses. We split the dataset into training, validation, and testing set with $8:1:1$ ratio.

# of Type	# of Training	TER	SEER	SER
1	1,000	0.00%	28.86%	48.53%
1	8,000	0.00%	8.71%	14.81%
3	15,000	0.00%	7.49%	12.73%
23	35,000	0.00%	8.46%	14.45%

Table III. Impacts of Training Datasets.

DNN model				Size	EER
N	Channel	Kernel	FCN		
5	[(512*4),1500]	[5,3,3,1,1]	512	4.5MB	2.43%
5	[(512*4),1500]	[5,3,3,1,1]	16	2.7MB	3.33%
3	**[(512*2),1500]**	**[5,3,1]**	**16**	**1.7MB**	**3.41%**
3	[(512*2),1500]	[1,2,1]	16	1.4MB	3.81%
3	[(128*2),256]	[1,2,1]	16	79.6KB	4.95%
3	[(32*2),64]	[1,2,1]	16	6.5KB	8.60%

Table IV. VUI response identification ablation study.

VUI Response Identification: We use EER, a well-known speaker identification metric, at which both acceptance and rejection errors are equal [43], to evaluate our VUI response identification method. Here we aim to identify the 10 voices (5 male and 5 female) on Google Assistant, along with a class for other non-VUI sound sources. Table IV summarizes the results. Since a small model degrades the performance significantly while a large model consumes more system overhead (Sec. IX-F), we empirically choose a medium size model of 1.7 MB model as our default model. As shown in Fig. 10, *our DNN model is able to accurately identify the different voice sources, with a maximum EER of less than* 6.73% *(*3.41% *on average)*.

VUI Response Private Entity Recognition: Unlike speech recognition metrics, such as Word Error Rate (WER) and Character Error Rate (CER), a more reasonable way to evaluate the private entity recognition within VUI response is to check whether StealthyIMU can understand the whole entity correctly. Therefore, we use the following 3 metrics: *i) Type Error Rate (TER)* which indicates whether StealthyIMU can recognize the type of the VUI response, *i.e.*, weather, calendar and navigation; *ii) Single Entity Error Rate (SEER)*. There may exist more than one entities in a single VUI response. SEER mainly takes a single entity as a unit to report the error rate; *iii) Sentence Error Rate (SER)* which counts an error-free recognition only if all the entities in a single VUI response are identified correctly.

We compare the performance of 3 different models to resolve the VUI response private entity recognition task, 1) "ASR+NLU": a cascaded approach with an ASR model to recognize the text from the MSS and an NLU model to extract the entity list from the text (Sec. II); 2) "SLU": StealthyIMU's SLU model without speech model knowledge distillation; 3) "SLU+KD": StealthyIMU's SLU model with speech model knowledge distillation. Table VI summarizes the results. The key problem of "ASR + NLU" is that the ASR model can only achieve 68.16% WER, which makes it hard for the follow-on NLU to extract the correct private entities. In contrast, SLU model directly optimizes the target of extracting the private entities from MSS. With the help of knowledge distillation from the teacher model, it becomes even more accurate. *On average, "SLU+KD" StealthyIMU achieves a* 0.00% *TER,* 8.46% *SEER, and* 14.45% *SER with a* 3.9 *MB model size.*

Next, we conduct a ablation study on the StealthyIMU SLU model by modifying the parameters of the Tokenizer,

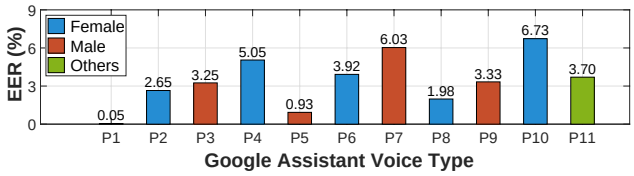


Figure 10. Google Assistant Voice Identification

Encoder and Decoder, as shown in Fig. V. We find that: *(i)* A subword-level tokenizer ($N_t = 50$) does not improve performance when the model size is small since it needs more parameters to learn the subword level characteristics. *(ii)* RNN is the most important block in our encoder. If we reduce the layers of the RNN blocks, the model performance will degrade significantly. *(iii)* The attention and hidden layer size impact the performance of the decoder. Based on these observations, we choose the model with 3.9 MB size as default model for VUI response private entity recognition (highlight in Table V).

Finally, we conduct experiments to evaluate the impacts of training and testing dataset. In Tab. III, we evaluate StealthyIMU in three different settings, *i.e.*, training and testing StealthyIMU by using *(i)* single type of voice command; *(ii)* 3 types of voice command with large-scale dataset (Weather, Reminders and Navigation App in Tab. II); *(iii)* all of 23 types of voice command. We find that *(i)* training and testing StealthyIMU with 3 types of voice commands achieve similar results as training/testing with 23 types of voice commands. *(ii)* if we use a single type of voice command to train the model, StealthyIMU requires a large-scale dataset to achieve high performance. In comparison, if we train the model with multiple types of voice commands, the scale of a specific type of voice command can be small. We believe that this is because a larger dataset with more diverse voice commands empowers the model to better generalize, so it is able to recognize the private entities from different formats of VUI responses.

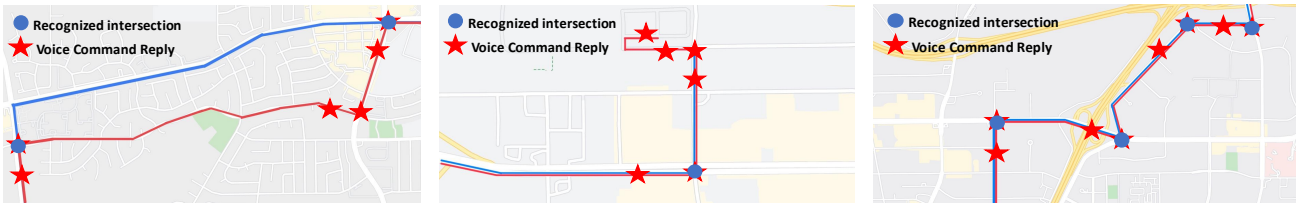
B. One-time Stealing

In the one-time stealing experiments, we use the same training and testing setting as in Sec. IX-A. A single model is trained to extract the private entities from a single VUI response across 23 types of voice commands. We select the model and parameters with the highest performance in Sec. IX-A. For the rest of the evaluation, we use this general DNN model as our default model to extract the private entities without training any new DNN models.

Tab. VII shows that *a single DNN model can accomplish one-time stealing with an average success rate of* 85.28% *even without contextual information*. Moreover, StealthyIMU achieve higher performance for structured VUI responses, *i.e.* "Weather", "Sunset & Sunrise", "AirCheck", "Clock", "Stock Update" *etc*. Although voice commands like "Reminders", "Media Alarm", "Hands Free Calling" and "Navigation App" has complicated response formats resulting in relatively lower success rate for one-time stealing, they can still achieve $> 70\%$ success rate. We believe that with more training data, they will achieve even higher success rate.

Token	Encoder (CDRNN)							Decoder (GRU + KeyValue Attention)				Model Size	TER	SEER	SER
	CNN Blocks			RNN Blocks		DNN Blocks									
N_t	N_c	K_c	C_c	N_r	H_r	N_{ed}	O_{ed}	N_d	H_d	A_d	E_d				
50	2	3	(128, 256)	4	1024	2	512	3	512	512	128	106.4 MB	0.00%	13.22%	21.31%
50	2	3	(64, 128)	4	256	2	256	3	256	256	64	9.1 MB	0.00%	9.65%	15.32%
100	2	3	(128, 256)	4	256	2	256	3	256	256	64	11.9 MB	0.00%	11.68%	20.01%
50	2	3	(64, 128)	4	128	2	256	3	256	256	64	4.2 MB	0.00%	10.85%	18.99%
50	2	3	(64, 128)	4	128	2	128	3	256	256	64	4.1 MB	0.00%	11.73%	19.96%
50	2	3	(64, 128)	2	128	2	128	3	256	256	64	3.3 MB	0.00%	17.91%	31.27%
50	2	3	(64, 128)	4	128	2	128	3	128	128	64	3.0 MB	0.00%	26.00%	46.40%
50	**1**	**3**	**(64, 128)**	**4**	**128**	**2**	**128**	**3**	**256**	**256**	**64**	**3.9 MB**	**0.00%**	**8.46%**	**14.45%**
50	1	3	(64, 128)	4	128	1	128	3	256	256	64	3.8 MB	0.00%	13.80%	24.60%
50	0	/	/	4	128	2	128	3	256	256	64	2.9 MB	0.00%	18.46%	34.85%

Table V. VUI Response Private Entity Recognition Ablation Study. N_t: # of tokens in tokenizer; N_c: # of CNN blocks; K_c: kernel size; C_c: CNN channels; N_r: # of RNN layers; H_r: # of RNN neurons; N_{ed}: # of DNN blocks; O_{ed}: # of DNN neurons; N_d: # of decoder layers; H_d: hidden size of decoder; A_d: attention dim of decoder; E_d: # of output neurons.



(a) Case 1 (b) Case 2 (c) Case 3

Figure 11. GPS trace recovery examples

	Model Size	TER	SEER	SER
ASR+NLU	26.5 MB	0%	46.45%	77.91%
SLU	3.8 MB	0%	14.76%	25.16%
SLU+KD	3.8 MB	0%	8.46%	14.45%

Table VI. SLU model training approach.

Type	Private Entity	TER	SEER	SER
Weather	Location	0.00%	3.05%	5.38%
Sunset & Sunrise	Location	0.00%	7.14%	13.97%
AirCheck	Location	0.00%	1.49%	3.33%
Clock	Location	0.00%	2.13%	3.18%
Reminders	Todo	0.00%	7.36%	13.21%
	Time	0.00%	15.25%	29.94%
Media Alarms	Todo	0.00%	8.24%	15.29%
	Time	0.00%	14.11%	26.50%
Stock Updates	Search	0.00%	7.33%	11.54%
Hands Free Calling	Contacts	0.00%	12.18%	22.64%
Navigation	Location	0.00%	2.19%	3.89%
Navigation App	GPS	0.00%	16.2%	26.79%
Others	/	0.31%	0.31%	0.31%
Overall		*0.00%*	*8.06%*	*14.45%*

Table VII. One-time stealing results.

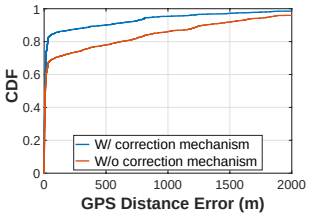


Figure 12. Short-term

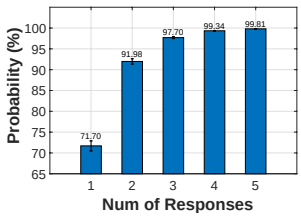


Figure 13. Long-term

C. Short-term Contextual Inference

We now proceed to verify the short-term continuous navigation voice and the GPS trace recovery algorithm. We collect a testing dataset in City A and make sure there is no overlap with the training dataset in Sec. IX-B. Then, we use the DNN model trained in Sec. IX-B to extract the private entities of each VUI response from the testing dataset. Finally, we apply the GPS trace recovery algorithm to the extracted private entities from consecutive VUI responses and compare the result with the ground truth GPS trace. To better illustrate our approach, we first show three examples of GPS trace recovery in Fig. 11. Then we evaluate the effectiveness of our route correction mechanism, and present the results of the end point localization.

In Case 1 of Fig. 11, our model only identifies 2 out of 3 intersections, and the shortest path between these two points deviates from the true path. If we could accurately identify the 3 intersections, then the path would be correct. Case 2 shows that if the final destination is on an unnamed road, our algorithm can only assign the last named road as the end point. Case 3 shows a perfect example of GPS trace recovery, where all the intersection points are accurately identified. These examples demonstrate that our recovery algorithm needs to accurately recognize the road names, based on which it can generate the final route through the shortest path between consecutive intersections on the map.

In order to quantify the actual performance of the algorithm, we perform a real-world field test in City A during daily driving and recover over 500-mile real routes. During the navigation process, we simultaneously record the results with and without the route correction mechanism. In terms of route coverage, the average route coverage rate without route correction is 52.46%, and increases to 62.87% after correction. Moreover, as shown in Fig. 12, 80% of the GPS distance errors are within 30 m after correction, in contrast to 63% before correction. In addition, we count the distance deviation between the recovered destination and the ground truth. According to our statistics, the min/average/max deviation is 3 m/133 m/420 m. The average deviation is large, because sometimes the destination is on an unnamed road, and we can only use the end point of the last named road as a replacement, as shown in Case 2 in Fig. 11.

D. Long-term Monitoring

Our long-term monitoring experiments focus on city name and home address identification.

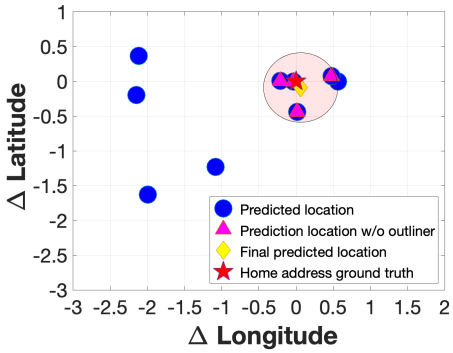


Figure 14. Long-term home address inference.

	Routes	Roads	Seg	SEER	SER
City A	300	419/441	5091	9.31%	19.77%
City B	150	559/1511	2794	16.25%	32.41%

Table VIII. Navigation voice for different cities.

For the city name extraction, we collect a testing dataset including "Weather", "Sunset & Sunrise", "AirCheck", "Clock" and "Navigation" voice commands over one months. Then, we perform private entity recognition on each VUI response in the testing set by using the default DNN model in Sec. IX-B. Then, we combine the results from multiple VUI responses by using the methods discussed in Sec. VI-C to demonstrate the scenario where StealthyIMU keeps monitoring the VUI responses over a few days. As shown in Fig. 13, as the victim repeats the inquiries over time, the probability that StealthyIMU correctly identifies the city name increases from 70% (1 inquiry) to above 98% (3+ inquiries).

For the home address identification, we collect a testing dataset containing 10-day routes back to home. Then, we follow the steps in Sec. IX-C to recover the GPS trace of these 10 routes, and mark the destination of the recovered GPS trace as the home address. Fig. 14 visualizes the results of home address inference. The ● represent the results of repeated attempts on home address inference, and ▲ represent the remaining GPS points after removing outliers. Finally, we use the centroid of these remaining GPS points as the address estimation. In this example, the deviation between the StealthyIMU estimation and the ground truth home destination is only 11 meters by combining 10 attempts of address estimation.

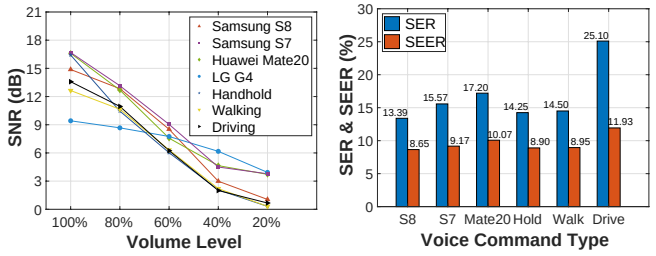
E. Generalization

In this section, we evaluate StealthyIMU generalization across different factors to see whether the model can work on a new scenario. By default, we use the model trained in Sec. IX-B as our baseline model, and collect additional testing datasets with different real-world scenarios.

Generalization across different sound volumes, smartphone models and sampling rate settings: Loudspeaker volume determines the vibration magnitude of the smartphone during a VUI response, and hence affects the SNR of the MSS. We evaluate 5 different volume levels as shown in Fig. 15(a). When the volume exceeds the default level of our experiments (80%), StealthyIMU achieves $>$ 10 dB SNR, and the SER (SEER) is smaller than 10% (15%). When the volume falls below 40%, the SER (SEER) is $>$ 80% (95%).

Phone	OS Version	Sampling Rate	SER	SEER
OnePlus	Android 11	440 Hz	13.85%	8.99%
Samsung S8	Android 9	400 Hz	13.39%	8.65%
Samsung S8	Android 9	200 Hz	62.69%	38.30%
Samsung S8	Android 9	100 Hz	84.01%	52.50%
Huawei Mate 20	Android 9	500 Hz	17.20%	10.07%
Samsung S7	Android 8	420 Hz	15.57%	9.17%

Table IX. Generalization across different smartphones and sampling rate



(a) SNR (b) SER & SEER

Figure 15. Generalization across different smartphones, volume levels, and motion artifacts. "Driving" achieves higher performance because it is evaluated by navigation data set only while others are evaluated by general dataset.

We further run StealthyIMU on 5 different smartphones, equipped with IMU sensors from different vendors and different sampling rate (100 $\sim$ 500 Hz). As shown in Fig. 15(a), smartphones released within 5 years, *i.e.*, OnePlus, Samsung S7, S8, Huawei Mate20, all support $>$ 400 Hz sampling rate which suffices for the StealthyIMU attack. Although the layout of the IMU sensor on the smartphone motherboard may vary, the signal SNR is similar among these smartphones. When the volume level is higher than 80%, StealthyIMU DNN model achieves an 15.39% SEER and 9.30% SER on average when trained with the mixed dataset collected from these three smartphones. On the other hand, the StealthyIMU attack becomes less effective when the MSS sampling rate falls below 200 Hz.

Generalization under different motion artifacts: We measure the impacts of interference from 3 motion artifacts: holding the smartphone in hand, walking, and driving with the smartphone on a car phone mount. Most of the noise introduced by human motion, like holding and walking, is low-frequency [27]. Thus, after applying a high-pass filter, the signal SNR remains high, while the SEER and SER are 10.3% and 16.8%–only slightly higher than the static case.

In the driving scenario, the main vibration interference comes from the car's body. The dominant noise frequency can be estimated by $f_n(t) = \frac{R(t)}{60} * (N_c/2)$, where $R(t)$ is the rotational speed of engine in revolutions per minute (RPM) and N_c is the cylinder number of the engine. For example, a car with a 4-cylinder engine generate the vibration noise of less than 100 Hz even at 3000 RPM (the RPM for normal driving is 1500 $\sim$ 2500 [44]). Therefore, to isolate the driving noise, we simply apply a high-pass filter with 100 Hz cut-off frequency and use 24 frequency bins within the frequency range of $(100, 250]$ Hz as the input features to retrain the DNN model. StealthyIMU maintains a high signal SNR of 12.56 dB, and low SER (SEER) of 25.10% (11.93%), when the victim smartphone is in a car with a 4-cylinder engine and speed ranging from 30 to 110 km/h.

	Mobile Data	Segment Storage	Segment Memory	Peak CPU	Power
1	100 KB/min	/	/	/	/
2	16 KB/Seg	16 KB/Seg	/	5%	36 mW
3	16 KB/Seg	/	16 KB/Seg	5%	35 mW
4	/	/	16 KB/Seg	5%	35 mW

Table X. Voice detection and segmentation overhead.

	Model Size	Peak CPU	APP Memory	Time (s/Seg)	Energy (mAh/Seg)
ID	79.6 KB	5%	4.1 MB	$9.8e-3$	$6.5e-3$
ID	1.7 MB	5%	7.4 MB	$53.3e-3$	$1.1e-3$
SLU	9.1 MB	13%	54.5 MB	4.60	0.65
SLU	3.9 MB	12%	34.5 MB	1.19	0.17

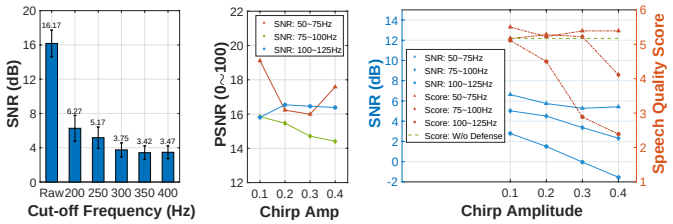
Table XI. On-device DNN overhead. ID: VUI response identification model; SLU: VUI response private entity recognition model.

Generalization across different cities: We perform a fake GPS test of StealthyIMU in two cities with different geographical layout (Sec. VIII-A). Note that StealthyIMU can only extract the GPS location for a specific city with the training dataset, and the DNN model needs to have the capability to extract the road names in a specific city. Therefore, we train the private entity recognition DNN models for city A and B respectively and evaluate the performance by using the corresponding model. This procedure is practical in real-world because the attacker can easily extract the city name of the victim's location by using long-term monitoring (Sec. VI-C) and select the corresponding model for further attack. As shown in Table VIII, although our dataset only covers 37% of the roads in City B, the SEER and SER of City B is still reasonably low (16.25% and 32.41% respectively), because even with such a small dataset, StealthyIMU can already recognize the main roads in the city, which will be frequently passed by users. By enriching our dataset, the accuracy can be further improved. We leave such incremental refinement for future work.

F. System Overhead Evaluation

We evaluate the system overhead and required permissions when processing StealthyIMU in cloud or on device. Based on our threat model (Sec. III), we summarize 4 deployment models of StealthyIMU: *1.* The malicious app steams the MSS to the cloud for processing; *2.* The malicious app detects the voice-associated signals on device and saves the segments in local storage, and then the segments will be uploaded to the adversaries' cloud for further processing; *3.* The process is similar to 2 except that the segments will be saved in app memory; *4.* StealthyIMU attack is deployed fully on device.

To compare the 4 options, we deploy the StealthyIMU attack on Samsung Galaxy S8 with Qualcomm Snapdragon 835 CPU, assuming that StealthyIMU is not able to access the GPU resources. We use Android Profiler [45] and app power monitor to measure the on-device overhead in terms of energy, CPU, memory, storage, and mobile data usage. Table X shows the results. Option 1 requires the "network" permission when the StealthyIMU app is running in the background and consumes 100 KB/min mobile data. In comparison, both option 2 and 3 need to upload the data to the cloud only when the voice is detected. The difference is that option 2 needs the "storage" permission whereas option 3 does not. Option 4 executes the entire StealthyIMU on device with zero permission. The peak CPU usage and power consumption of



(a) f_c v.s. SNR (b) PSNR (c) SNR v.s. Speech Quality

Figure 16. Predistortion Speech Defense

Chirp freq (Hz)	Amp	Comfort	Intelligibility	Noise	Overall
/	0	5.5	5.17	4.83	5.17
50∼75	0.1	5.33	6.00	5.17	5.50
50∼75	0.2	4.83	5.83	5.00	5.22
50∼75	0.3	5.17	6.00	5.00	5.39
50∼75	0.4	5.17	5.67	5.33	5.39
75∼100	0.1	4.67	6.00	4.83	5.17
75∼100	0.2	4.83	5.67	5.33	5.28
75∼100	0.3	5.17	5.83	4.67	5.22
75∼100	0.4	3.83	5.17	3.33	4.11
100∼125	**0.1**	**4.67**	**5.50**	**5.17**	**5.11**
100∼125	0.2	4.33	5.33	3.83	4.50
100∼125	0.3	2.50	3.83	2.33	2.89
100∼125	0.4	1.83	3.33	2.00	2.39

Table XII. Defense Speech Quality Subjective Assessment

voice detection and segmentation are less than 5% and 46 mW respectively, which means that it only consumes 5.6% battery for 24 hours on a typical smartphone (3000 mAh, 3.7 V).

Table XI shows the on-device DNN overhead. The input of our DNN is the voice-associated MSS segments, so we use seconds per segment and mAh per segment to measure the time and energy consumption of our models. Medium sized DNN models, *i.e.*, identification model with 1.7 MB model size and SLU with 3.9 MB model size, are more feasible for the on-device attack because the performance is close to the large models (Sec. IX-A) while the overhead is significantly lower. The app memory consumption is less than 42 MB in total and the peak CPU usage is less than 13%. These DNN models can recognize 176 voice-associated segments with less than 1% battery consumption. Overall, the behavior of the zero-permission on-device StealthyIMU is unlikely to be distinguishable from an innocuous app.

G. Defense Evaluation

We use three metrics to evaluate the defense capability of the proposed speech predistortion defense: 1) SNR, 2) PSNR, and 3) DNN model SEER and SER. As shown in Fig. 16(a), the high pass filter mechanism can reduce the SNR of the MSS by 12.7 dB. When the cut-off frequency f_c exceeds 350 Hz, the SNR will not decrease significantly anymore since this is close to the highest sampling rate (500 Hz) and the harmonics of higher frequencies start to emerge. Further augmented with chirp distortion, our defense mechanism reduces the PSNR down to 16 when the chirp frequency is set to 100 ∼125 Hz and amplitude 0.1. This implies the structural features within the spectrogram are substantially corrupted, since intelligible speech requires $25 \sim 30$ PSNR [46].

Next, we investigate the trade-off between defense capability and speech quality. We conducted a subjective assessment to evaluate the speech quality after our speech predistortion

defense. We recruited 30 volunteers (23 males and 7 female with ages in the range of 19 to 27) to assess the speech quality. In each study, we first randomly select a speech signal with a single VUI response with 12 different predistortion defense parameters and a raw signal without predistortion defense. And then the volunteers are asked to listen to the selected speech signals transmitted by 15 different smartphones without any prior knowledge of the signals. After listening to the speech signals, three metrics are assessed through a questionnaire with $1 \sim 7$ rating [47] for each question: *(i)* Comfort: Do you feel comfortable with that speech? *(ii)* Intelligibility: Can you understand that speech? *(iii)* Quality: Can you hear noise in that speech signals? Each volunteer will assess all of 13 different speech signals. Table XII shows the results of our subjective assessment. Fig. 16(c) shows the mean values of the subjective metrics. We observe that, *with the high pass filter and the default chirp frequency band* $100 \sim 125$ *Hz and chirp amplitude* 0.1 *in the predistortion, the speech quality score is* 5.11*–close to that without predistortion (*5.17*), whereas the attacker's SNR and PSNR drop sharply (*2.78 *dB and* 15.81*) below the threshold for intelligible speech.*

We further verify whether an attacker can circumvent the predistortion defense by using the predistorted MSS to train its SLU model. We first apply predistortion to our MSS dataset, and then use the SLU model configurations with the best performance to train and test the dataset. The result shows 95.06% SER and 89.17% SEER, which means that *the predistortion defense is sufficient to prevent the StealthyIMU from recognizing the private intents*. The attacker cannot reverse the speech predistortion from the MSS, even if it knows how the predistortion is applied.

X. Conclusion

We have demonstrated the feasibility and effectiveness of StealthyIMU, a new threat that allows a zero-permission app to steal private information from VUI responses on a smartphone. The attack surface lies in a side channel, where in a motion sensor "overhears" the low-frequency vibration from the co-located loudspeaker. Although word-by-word transcription of general speech is challenging [19], we leverage the deterministic features of the machine-rendered VUI responses, and design a speech detection and understanding model to extract the private information. We further optimize the model and limit its resource usage, so it is indistinguishable from an innocuous app. Our case studies show that StealthyIMU can accurately steal crucial permission-protected private information, such as contacts, search history, calendar, home address, and GPS routes, from popular VUIs such as Google Assistant and Google Map. We further develop effective defense mechanisms which can help VUI vendors remove the vulnerability.

Acknowledgment

We would like to thank the anonymous reviewers for their valuable comments. This work is partially supported by the NSF under Grants CNS-1901048, CNS-1925767, and CNS-2128588.

References

[1] Z. Ba, T. Zheng, X. Zhang, Z. Qin, B. Li, X. Liu, and K. Ren, "Learning-based practical smartphone eavesdropping with built-in accelerometer," in *Proceedings of NDSS*, 2020.

[2] S. A. Anand, C. Wang, J. Liu, N. Saxena, and Y. Chen, "Spearphone: a lightweight speech privacy exploit via accelerometer-sensed reverberations from smartphone loudspeakers," in *Proceedings of ACM WiSec*, 2021.

[3] S. Dey, N. Roy, W. Xu, R. R. Choudhury, and S. Nelakuditi, "Accelprint: Imperfections of accelerometers make smartphones trackable.," in *NDSS*, 2014.

[4] L. Zhang, P. H. Pathak, M. Wu, Y. Zhao, and P. Mohapatra, "Accelword: Energy efficient hotword detection through accelerometer," in *Proceedings of ACM MobiSys*, 2015.

[5] C. Shi, X. Xu, T. Zhang, P. Walker, Y. Wu, J. Liu, N. Saxena, Y. Chen, and J. Yu, "Face-mic: inferring live speech and speaker identity via subtle facial dynamics captured by ar/vr motion sensors," in *Proceedings of ACM MobiCom*, 2021.

[6] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, "Accessory: password inference using accelerometers on smartphones," in *Proceedings of ACM Workshop on Mobile Computing Systems & Applications*, 2012.

[7] A. J. Aviv, B. Sapp, M. Blaze, and J. M. Smith, "Practicality of accelerometer side channels on smartphones," in *Proceedings of ACSAC*, 2012.

[8] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury, "Tapprints: your finger taps have fingerprints," in *Proceedings of ACM MobiSys*, 2012.

[9] H. Wang, T. T.-T. Lai, and R. Roy Choudhury, "Mole: Motion leaks through smartwatch sensors," in *Proceedings of ACM MobiCom*, 2015.

[10] X. Liu, Z. Zhou, W. Diao, Z. Li, and K. Zhang, "When good becomes evil: Keystroke inference with smartwatch," in *Proceedings of ACM CCS*, 2015.

[11] C. Wang, X. Guo, Y. Wang, Y. Chen, and B. Liu, "Friend or foe? your wearable devices reveal your personal pin," in *Proceedings of ACM AsiaCCS*, 2016.

[12] C. X. Lu, B. Du, H. Wen, S. Wang, A. Markham, I. Martinovic, Y. Shen, and N. Trigoni, "Snoopy: Sniffing your smartwatch passwords via deep sequence learning," *Proceedings of ACM IMWUT (UbiComp)*, 2018.

[13] J. Han, E. Owusu, L. T. Nguyen, A. Perrig, and J. Zhang, "Accomplice: Location inference using accelerometers on smartphones," in *Proceedings of IEEE COMSNETS)*, 2012.

[14] S. Narain, T. D. Vo-Huu, K. Block, and G. Noubir, "Inferring user routes and locations using zero-permission mobile sensors," in *2016 IEEE S&P*, pp. 397–413, IEEE, 2016.

[15] J. Hua, Z. Shen, and S. Zhong, "We can track you if you take the metro: Tracking metro riders using accelerometers on smartphones," *IEEE Transactions on Information Forensics and Security*, 2016.

[16] A. Mosenia, X. Dai, P. Mittal, and N. K. Jha, "Pinme: Tracking a smartphone user around the world," *IEEE Transactions on Multi-Scale Computing Systems*, 2017.

[17] Y. Michalevsky, A. Schulman, G. A. Veerapandian, D. Boneh, and G. Nakibly, "Powerspy: Location tracking using mobile device power analysis," in *Proceedings of USENIX Security*, 2015.

[18] Y. Michalevsky, D. Boneh, and G. Nakibly, "Gyrophone: Recognizing speech from gyroscope signals," in *Proceedings of USENIX Security Symposium (USENIX Security)*, 2014.

[19] S. A. Anand and N. Saxena, "Speechless: Analyzing the threat to speech privacy from smartphone motion sensors," in *Proceedings of IEEE Symposium on Security and Privacy (S&P)*, 2018.

[20] S. A. Anand, C. Wang, J. Liu, N. Saxena, and Y. Chen, "Spearphone: A speech privacy exploit via accelerometer-sensed reverberations from smartphone loudspeakers," *arXiv preprint arXiv:1907.05972*, 2019.

[21] G. Tur and R. De Mori, *Spoken language understanding: Systems for extracting semantic information from speech*. John Wiley & Sons, 2011.

[22] D. Serdyuk, Y. Wang, C. Fuegen, A. Kumar, B. Liu, and Y. Bengio, "Towards end-to-end spoken language understanding," in *Proceedings of IEEE ICASSP*, 2018.

[23] A. Coucke, A. Saade, A. Ball, T. Bluche, A. Caulier, D. Leroy, C. Doumouro, T. Gisselbrecht, F. Caltagirone, T. Lavril, *et al.*, "Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces," *arXiv:1805.10190*, 2018.

[24] A. Caubrière, S. Ghannay, N. Tomashenko, R. De Mori, A. Laurent, E. Morin, and Y. Estève, "Error analysis applied to end-to-end spoken language understanding," in *Proceedings of IEEE ICASSP*, 2020.

[25] Google, "Android permission API reference," 2019. https://developer.android.com/reference/android/Manifest.permission/.

[26] Q. Li, J. A. Stankovic, M. A. Hanson, A. T. Barth, J. Lach, and G. Zhou, "Accurate, fast fall detection using gyroscopes and accelerometer-derived posture information," in *Proceedings of IEEE International Workshop on Wearable and Implantable Body Sensor Networks*, 2009.

[27] R. Khusainov, D. Azzi, I. E. Achumba, and S. D. Bersch, "Real-time human ambulation, activity, and physiological monitoring: Taxonomy of issues, techniques, applications, challenges and limitations," *Sensors*, 2013.

[28] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, "X-vectors: Robust dnn embeddings for speaker recognition," in *Proceedings of IEEE ICASSP*, 2018.

[29] D. Snyder, D. Garcia-Romero, D. Povey, and S. Khudanpur, "Deep neural network embeddings for text-independent speaker verification.," in *Proceedings of Interspeech*, 2017.

[30] K. Okabe, T. Koshinaka, and K. Shinoda, "Attentive statistics pooling for deep speaker embedding," in *Proceedings of Interspeech*, 2018.

[31] A. Nagrani, J. S. Chung, and A. Zisserman, "Voxceleb: a large-scale speaker identification dataset," *Proceedings of Interspeech*, 2017.

[32] M. Zeineldeen, A. Zeyer, W. Zhou, T. Ng, R. Schlüter, and H. Ney, "A systematic comparison of grapheme-based vs. phoneme-based label units for encoder-decoder-attention models," *arXiv:2005.09336*, 2020.

[33] T. Kudo and J. Richardson in *Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing*, 2018.

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of NeurIPS*, 2017.

[35] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of NeurIPS*, 2014.

[36] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *Proceedings of IEEE ICASSP*, 2015.

[37] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *Proceedings of NeurIPS Deep Learning Workshop*, 2014.

[38] M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *IEEE Pervasive computing*, 2008.

[39] K. Sun, C. Chen, and X. Zhang, ""alexa, stop spying on me!" speech privacy protection against voice assistants," in *Proceedings of ACM SenSys*, 2020.

[40] S. Schuster, S. Gupta, R. Shah, and M. Lewis, "Cross-lingual transfer learning for multilingual task oriented dialog," *Proceedings of NAACL*, 2019.

[41] N. Verma, *Mobile Test Automation With Appium*. Packt Publishing Ltd, 2017.

[42] S. Wiesler, A. Richard, R. Schlüter, and H. Ney, "Mean-normalized stochastic gradient for large-scale deep learning," in *Proceedings of IEEE ICASSP*, 2014.

[43] J. Oglesby, "What's in a number? moving beyond the equal error rate," *Speech communication*, 1995.

[44] autoblog, "How to Monitor Your RPM Gauge to Get the Best Performance Out of Your Car," 2016. https://www.autoblog.com/2016/04/14/how-to-monitor-your-rpm-gauge-to-get-the-best-performance-out-of/.

[45] "Android Profiler," 2022. https://developer.android.com/studio/profile.

[46] S. Zhu, C. Zhang, and X. Zhang, "Automating Visual Privacy Protection Using a Smart LED," in *Proceedings of ACM MobiCom*, 2017.

[47] G. A. Miller, "The magical number seven, plus or minus two: Some limits on our capacity for processing information.," *Psychological review*, 1956.