Received September 13, 2019, accepted October 1, 2019, date of publication October 7, 2019, date of current version November 1, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2945791

Using AI to Attack VA: A Stealthy Spyware Against Voice Assistances in Smart Phones

RONGJUNCHEN ZHANG[1], XIAO CHEN[1], SHENG WEN[2], XI ZHENG[3], AND YONG DING[2,4]

[1]Faculty of Science, Engineering and Technology, Swinburne University of Technology, Hawthorn, VIC 3122, Australia
[2]Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen 518066, China
[3]Faculty of Science and Engineering, Macquarie University, Macquarie Park, NSW 2109, Australia
[4]School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin Shi 541004, China

Corresponding author: Sheng Wen (swen.works@gmail.com)

This work was supported in part by the National Natural Science Foundation of China (Source Identification and Risk Evaluation for Malicious Information based on Sparsely Observed Data, from January 2018 to December 2021) under Grant 61772148.

ABSTRACT Voice Assistants (VAs) are increasingly popular for human-computer interaction (HCI) smartphones. To help users automatically conduct various tasks, these tools usually come with high privileges and are able to access sensitive system resources. A comprised VA is a stepping stone for attackers to hack into users' phones. Prior work has experimentally demonstrated that VAs can be a promising attack point for HCI tools. However, the state-of-the-art approaches require ad-hoc mechanisms to activate VAs that are non-trivial to trigger in practice and are usually limited to specific mobile platforms. To mitigate the limitations faced by the state-of-the-art, we propose a novel attack approach, namely `Vaspy`, which crafts the users' "activation voice" by silently listening to users' phone calls. Once the activation voice is formed, Vaspy can select a suitable occasion to launch an attack. Vaspy embodies a machine learning model that learns suitable attacking times to prevent the attack from being noticed by the user. We implement a proof-of-concept spyware and test it on a range of popular Android phones. The experimental results demonstrate that this approach can silently craft the activation voice of the users and launch attacks. In the wrong hands, a technique like Vaspy can enable automated attacks to HCI tools. By raising awareness, we urge the community and manufacturers to revisit the risks of VAs and subsequently revise the activation logic to be resilient to the style of attacks proposed in this work.

INDEX TERMS Voice assistant, smartphone, android, software security, systems security.

I. INTRODUCTION

Voice assistants (VAs) have been widely used in smartphones, typically as human-computer interaction (HCI) mechanisms for device control and identity authentication. Popular examples from the market include Amazon Alexa [1], Samsung Bixby [2], Google Assistant [3], and Apple Siri [4]. Because human-beings are able to speak about 150 words per minute, which is much faster than typing, *e.g.*, roughly 40 words per minute on average, VAs are very useful to transform human speech into machine-actionable commands. This creates an easy-to-use design of smartphones, especially for those that need lots of inputs or for scenarios where 'hands-free' is mandatory (*e.g.*, making phone calls when driving). In order to support broad functionalities via voice, *e.g.*, sending text messages, making phone calls, browsing the Internet, playing music/videos, *etc.*, VAs are usually granted high-level privileges including dangerous permissions [5] (*e.g.*, `ACCESS_COARSE_LOCATION`, `READ_CONTACTS`).

Unfortunately, the VA technique is a double-edged sword. They not only bring great convenience to smartphone users, but also offer a backdoor for hackers to gain entrance into the mobile systems [6]. Hackers can take advantage of VAs' required high privilege in accessing various applications and system services to steal users' private information like locations and device IDs [3], control smart home devices [7], forge emails, or even transfer money [8], *etc*. For example, after activating the Google Assistant with the keywords "`OK Google`", a hacker can further manipulate an episode of attacking voice that cheats the smartphone to send the user's location to a specific number via SMS with commands such as "`send my location to 12345678`" [3]. Given a list of VA-enabled functions [9], we can identify many potential attacks against users' smartphones [10].

Prior work has already demonstrated the feasibility of attacking smartphones via VAs [1], [3], [11], [12]. The key

The associate editor coordinating the review of this manuscript and approving it for publication was Shui Yu.

to the successes of the approaches is to activate VAs in a stealthy manner. For example, W. Diao *et al.* [3] and Alepis and Patsakis [1] utilise the Android inter-component communication (ICC) to wake up the VA. To be stealthy, they propose to launch attacks when smartphones are unattended or in the early morning (*e.g.*, 3 am). However, this approach requires to call a specific API ('`ACTION_VOICE_SEARCH_HANDS_FREE`'), which is only available in Google Assistant. This excludes the use of the approach in some brands like Huawei and Xiaomi, which provide custom VAs other than Google Assistant. Zhang *et al*. [12] propose using inaudible ultrasound to activate VAs. The attacking commands are undetectable by users but can be recognised by VAs on smartphones. However, this approach needs a special ultrasound generator on-site, which is not practical in the real world. There is another work under the same umbrella. Carlini *et al.* [11] apply adversarial machine learning technique to manipulate attacking sounds against voice recognition systems. This approach requires the hackers to have physical access to the targeting smartphones and run sound crafting processes iteratively. This premise is also impractical in most real-world scenarios.

In this paper, we propose a novel and practical stealthy attacking approach against voice assistants in Android phones, named `Vaspy`. It learns from the user's normal dialogue to craft the activation voice to the VA and leverages the built-in speaker to play and activate the VA. To be stealthy, the attack is triggered only at moments when the smartphone user is most likely to overlook the occurrence of activation voice. The idea of Vaspy comes from two practical facts: 1) the built-in speaker can be used to activate the VA of a phone [1]; and 2) the ringtone of a phone can be easily neglected by a user in a noisy environment.

We develop a proof-of-concept spyware based on Vaspy. The spyware disguises itself as a popular microphone controlled game to increase the chance of successful delivery to targeting Android phones.[1] The spyware records in/outbound calls and synthesises the activation keywords (*e.g.*, '`OK Google`') using speech recognition and voice cloning [14] techniques. This operation is necessary as state-of-the-art VAs are resilient to unauthenticated voiceprints. The proof-of-concept spyware sheds light on two advantages of Vaspy: 1) since the attacking process only makes use of a common component in an Android phone (*e.g.*, the built-in speaker), Vaspy can be applied to most off-the-shelf Android phones that have built-in VAs; this breaks the limitations in prior work, which either requires a special equipment [11], [12] or can only be applied to Google Assistant [1], [3]; 2) Vaspy can employ machine learning techniques to analyse data collected from various on-board sensors; this helps Vaspy identify the optimal attacking time, making it stealthier compared to prior work [1], [3].

[1]This is only an example for delivery. There are many other social engineering methods to be used in the real world, *e.g.*, [13].

Vaspy can be very dangerous to smartphone users, not only due to its stealthiness, but also because of its resilience to state-of-art anti-virus tools. We test the proof-of-concept spyware on VirusTotal [15], a widely adopted industrial anti-virus platform. We also test the spyware on three state-of-the-art learning-based Android malware detectors, namely Drebin [16], DroidAPIMiner [17], and MaMaDroid [18]. Results indicate that the spyware based on Vaspy can evade their detection. In fact, Vaspy seldom invokes sensitive APIs and uses the VA as a puppet to carry out malicious activities, making it resilient to those anti-virus tools.

We summarise the contributions of this paper as follows.

- We propose a novel attacking approach called Vaspy, which can stealthily hack into Android phones via built-in VAs without users' awareness.
- We designed a context-aware module in Vaspy, making it stealthier compared to prior work. This module provides intelligent environment detection to identify the optimal time to launch attack, based on the data collected from various on-board sensors.
- We develop a proof-of-concept spyware based on Vaspy to evaluate the attack in a real-world empirical study. The empirical results show that users cannot detect the spyware and the spyware does not affect the performance of Android phones significantly. We also find that the spyware is resilient to typical anti-virus tools from both industry and academia.

The rest of this paper is organised as follows. Section II presents related works. Section III provides the details of the attacking model in Vaspy. Section IV-A demonstrates the feasibility of Vaspy through a proof-of-concept spyware. The evaluation is presented in Section V, followed by a discussion of some open issues in Section VI. Section VII concludes this paper.

II. RELATED WORK

A. ATTACKS TO SMARTPHONE VA

There are a few existing work designed to attack VAs. For example, Alepis and Patsakis [1] Diao *et al.* [3] proposed an attacking method that made use of Android inter-component communication mechanism and built-in speaker. To be stealthy, Diao *et al*. [3] designed the attack to be triggered at 3 `am`, a time when smartphones were expected to be unattended (*e.g.*, users sleeping). A similar model to make the attack stealthy was adopted in E. Alepis *et al*.'s work [1]. However, these attacks require a specific API (Intent: '`ACTION_VOICE_SEARCH_HANDS_FREE`'), which was only available in Google Assistant. This limits the use of their proposed attacking methods, *e.g.*, considering devices like Huawei's Xiao Yi and Xiaomi's Xiao Ai, which provide custom VAs for users. In addition, the stealthiness of the above methods is not complete. For example, the volume of activation voice (*e.g.*, 55±3 dB claimed in Table 4 of [3]) may be loud enough to wake the user, considering the quiet environment in the early morning [19].

There are some other attacking methods that focused on crafting special audio that could be recognised by smartphone VAs but not heard by human-beings [11], [12]. For example, the idea of Carlini *et al*. [11] was to obfuscate raw attack audio and make it sound like a noise. Based on adversarial machine learning techniques [20], [21], the deliberately crafted audio could be recognised by smartphone VAs but was neglected by smartphone users as incomprehensible noise. In another example, Zhang *et al*. proposed using ultrasound [12], as its frequency is higher than the upper audible limit of human hearing. However, the approach of Carlini *et al*. [11] requires access to the targeting voice recognition model as either a black-box or a white-box, in order to run audio crafting processes iteratively. Moreover, the approach of Zhang *et al*. requires a special instrument (*e.g.*, ultrasound generator) [12]. Both premises are impractical in most real-world scenarios.

There are also some works that specifically studied the attacks against speech recognition systems (*note*: a key part in VA) [22]–[24]. For example, Yuan *et al.* [24] embedded voice commands into a song that can be recognised as a complete sentence by the speech recognition system. Schönherr *et al.* [23] manipulated adversarial examples against speech recognition systems by crafting special audio signals based on psycho-acoustic hiding technique. Kumar *et al.* [22] explored interpretation errors made by Amazon Alexa and found that Amazon Alexa could make some permanent systematic errors. All these works focus on audio processing for attacks. However, in the proposed Vaspy, we mainly focus on the stealthier attacking behaviours such as identifying suitable attack time and making it imperceptible to users. The ideas of the above works can also be borrowed and integrated into our Vaspy to expand the attack range.

B. CONTEXT-AWARENESS BASED ON SMARTPHONE SENSORS

The success of Vaspy relies on activating VAs in a stealthy manner. This in turn relies on context-awareness that identifies the optimal attacking time according to the data collected from the smartphone's on-board sensors (*e.g.*, accelerometer, gyroscope, and ambient light sensor). In this subsection, we analyse similar works that also adopted context-awareness based on on-board sensors.

D. Silva *et al*. adopted a series of sensors in a smart home to predict human activities [25]. J. Wiese *et al*. collected sensor data to analyse where people keep their smartphones [26]. They achieved an 85% successful rate in determining if a smart phone was in a bag, in a pocket, out, or in hand. Liu *et al*. [27] proposed recognising PINs when users input them by keyboard to smart watches. They used the accelerometer to capture user's hand movement, and achieved high accuracy in keystroke inference. In another work, user's typing pattern was learned via accelerometer readings [28]. These patterns were then used to infer user's typing on the screen. Moreover, J. Ho *et al*. proposed a context-awareness algorithm that determined when and what information to present would not make flawless decisions on mobile devices with heavy communication traffic [29]. We can find many similar applications of context-awareness based on smartphone sensors, *e.g.*, [30]–[32].

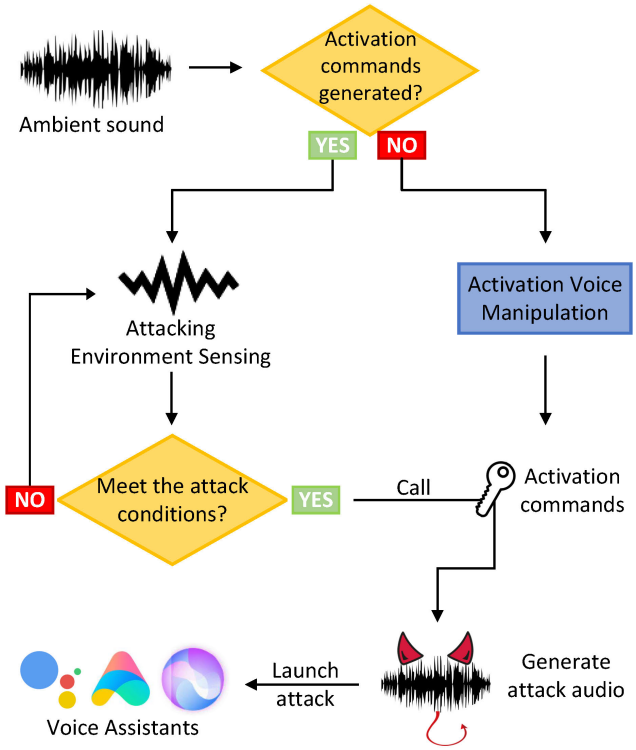


FIGURE 1. The workflow of an example spyware based on Vaspy. Ambient sound (*e.g.* Incoming/outgoing calls) is monitored and recorded, and the activation voice is then synthesised. User's environment is monitored by built-in sensors to determine a suitable attacking occasion. When launching the attack, text commands can be retrieved from Firebase [33] and generated attack audio by a built-in Text-to-Speech (TTS) module in the smartphone.

Similar to prior work, Vaspy also uses context-awareness based on smartphone sensors. In this area, we reckon that there is no superiority among different context-awareness methods. Vaspy just integrates those that can increase the chance of successful attacking. The particular approach may be different when Vaspy is implemented in various proof-of-concept scenarios.

III. ATTACKING MODEL: VASPY

The workflow of Vaspy is shown in Figure 1. Vaspy's attacking approach includes two modules: 1) Activation Voice Manipulation and 2) Attacking Environment Sensing. The first module synthesises the commands (*e.g.*, 'OK Google') that are required to activate the VA. Because most popular VAs can differentiate the voice of genuine smartphone owners based on artificial intelligence technologies [34], the activation voice in Vaspy will be manipulated based on the targeted users' own voice. This will ensure the success in activating smartphone VAs.

There are mainly two approaches available for synthesising activation voice: 1) using users' voice recording to clone an

activation voice [14]; and 2) extracting an activation voice form users' voice recordings. For the first approach, we can adopt voice cloning method [14] based on multi-speaker generative modelling [35] to generate the activation voice by a few users' own voice recordings. The method provides a trained multi-speaker model (fine-tuning) that takes a few audio-text pairs as input to simulate new speaker. This approach requires a text input to encode the cloned voice. Alternatively, the second approach adopts speech recognition techniques/tools such as Recurrent Neural Network (RNN) [36] to retrieve/synthesise the vocal pieces of those special words from users' own voice. This approach has been widely used in some commercial systems such as IBM Watson [37]. In Section IV-A, we implement an RNN-based method to synthesise users' voice in our proof-of-concept spyware, but alternative techniques/tools can also be integrated to Vaspy. In our implementation, the vocal corpus of special words can help craft the activation voice, *e.g.*, 'OK' plus 'Google' producing 'OK Google' as a whole activation voice piece for Google Assistant. However, it can be very challenging when the targeted user seldom speaks these special words. In this case, Vaspy will synthesise the vocal pieces of the special words from syllables captured from users' voice [38], *e.g.*, the first syllable of 'good' and the second syllable of 'single' can be concatenated to pronounce 'google'.

Once the activation commands are crafted, the second module will collect environment data such as light levels, noise levels, and motion states, via on-board sensors. Vaspy introduces machine learning techniques to decide an optimal time to launch the attack in a stealthy manner. The correctness of Vaspy's decisions is determined by the volume and quality of the contextual data collected to access the attacking environment. After the second module identifies a suitable attacking time, the synthesised activation voice is played, followed by prepared attacking commands (*e.g.*, "send my location to 123456"), causing harm to the targeted smartphone user. After the activation, successive attacking commands can be easily delivered to the VAs to control the compromised phone.

IV. PROOF-OF-CONCEPT: A SPYWARE

A. ACTIVATION VOICE MANIPULATION

We implement a proof-of-concept spyware in Android to evaluate Vaspy in a series of real-world scenarios. The spyware disguises itself as a microphone-controlled game. When a user starts playing the game, Vaspy will be activated in the background and stay active even if the game app is terminated.

Once launched, Vaspy registers itself as a foreground service[2] that monitors phone call status. When there is an incoming or outgoing call, Vaspy starts recording the audio from microphone. An audio clip is saved every 30 seconds. It will be processed by the Activation Voice Manipulation module and then be deleted immediately to release the storage. The recording process stops when either the phone call ends or the activation keyword is successfully synthesised.

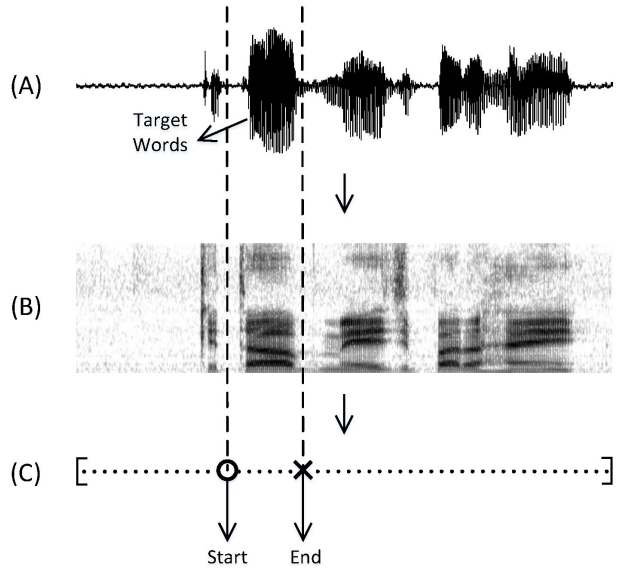


FIGURE 2. RNN training data pre-processing. (A) raw audio signal as input, which contains the activation keywords; (B) spectrum's converted from raw audio signal; (C) a matrix that contains labeled starting and ending frames of the activation keywords.

We implement a RNN-based voice synthesis model[3] in our proof-of-concept spyware. The RNN model is trained with audio clips containing both positive words (*i.e.*, activation keywords) and negative words (*i.e.*, non-activation words). The training clips are synthesised by human voices and background noises. Human voices are collected from 10 participants (5 males and 5 females). Fifty audio clips with positive words and fifty with negative words are collected, and then synthesised with different background noises. Eventually, 5, 000 training audio clips (13.9 hours in length) are generated. Figure 2 illustrates the process of preparing the training samples. Audio signals are converted into spectrograms which represent the spectrum of frequencies of the signals. Starting and ending frames of each activation keyword are labeled in the audio clips. The RNN is trained to extract activation words from audio clips. We implement the Gate Recurrent Unit as the core unit of our RNN [40]. There are 4, 500 and 500 audio clips used in training and testing, respectively. The accuracy on the testing set is 93.4%.

Note that in our prototype implementation, recorded audio clips must contain the activation keywords. However, this limitation can be removed by implementing voice cloning technique [14], which requires only a few voice recordings of arbitrary contents from the targeting user.

B. ATTACKING ENVIRONMENT SENSING

The Attacking Environment Sensing (AES) module determines the optimal attacking time based on the environment

[2]Android 9 disables background services from accessing user input and sensor data. Therefore, we use foreground service and hide the notification icon by making it transparent. [39]

[3]The trained models are perform on server to occupy local resources as less as possible.

data collected by the sensors. In particular, we extract the *movement intensity* features from accelerometer readings and the features of *environment variables* from microphone and light sensors readings. Since smartphones do not have built-in noise sensors, noise levels in decibel are calculated from the amplitude of the ambient sound that we gathered from microphone, according to $\mathbf{L_{dB}} = 10\,lg\left(\frac{A_1}{A_0}\right)^2$ wherein A_1 is the amplitude of the recorded sound, and A_0 is a standard amplitude that is usually set to one. The *movement intensity* features and *environment variables* features will be handled by the AES module to determine the attacking success rate. We treat this environment sensing problem as a classification problem, but there are only two results: attacking is successful or failed. The goal of AES module is to determine how likely an attacking will be successful instead of recognise a specific scenario.

Movement intensity features describe an overall perspective of human behaviour state. We divide human behaviours into a series of states, including 1) the definite motion state, 2) the definite stationary state, and the relative motion-stationary state. The definite motion state indicates significant fluctuation on sensor readings. The definite stationary state shows consistent sensor readings. The sharp difference of readings between the definite motion state and the definite stationary state allows the classification model to recognise these behaviours with high accuracy. However, the activities that do not show an apparent fluctuation may confuse the classification model. Therefore, to increase the classification accuracy, we define an intermediate, *i.e.*, a relative motion-stationary state, by which most of the confusing activities can be classified accurately. In this prototype, we use Random Forest as our classification model because RF does not directly output class labels but instead computes probabilities. We assign labels to the instances according to whether the probabilities of RF exceed a certain threshold. We label the motion state with the probability of over 60% and less than 40% as a definite motion state and a definite stationary state, respectively. We also label the motion state with the probability between 40% to 60% as a relative motion-stationary state. As the *movement intensity* features are categorical data, machine learning based algorithms cannot work with them directly. Therefore, we convert all the *movement intensity* features to numerical values using one-hot encoding. The definite motion state has been encoded to $[0, 1]$, the definite stationary state has been encoded to $[1, 0]$ and the relative motion-stationary state has been encoded to $[1, 1]$. The collected data is then fed into machine learning algorithm to train the AES model.

C. POST ATTACKS AND SPYWARE DELIVERY

Once the AES module determines to launch the attack, the synthesised activation voice is played via the speaker on the victim's phone. Meanwhile, the attacking commands, which are in text format, are dynamically fetched from Firebase [1] and played via the smart phone's speaker using Android built-in Text-to-speech (TTS) service. Attackers can then manipulate the voice assistant to further conduct malicious activities such as leaking private information, sending malicious SMS/email, *etc*.

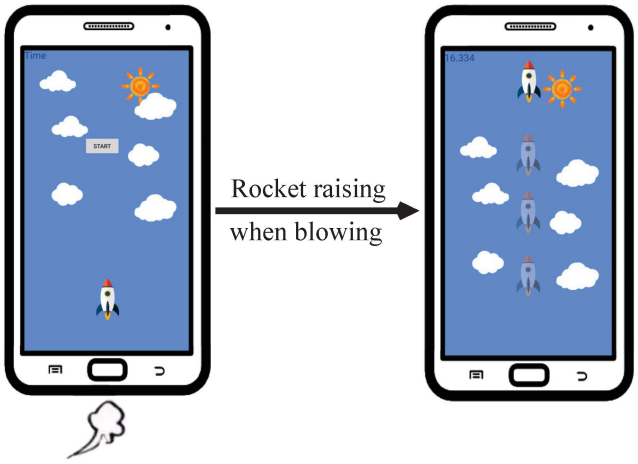


FIGURE 3. The snapshot of the proof-of-concept spyware. After player clicking start button, the rocket will raise when player blows or scream to the microphone. The rising speed depends on the volume of sound that the microphone receives.

Vaspy is able to utilise VA as an attacking tool which can bypass many permissions (*e.g.*, `SEND_SMS`). Nevertheless, three fundamental permissions are required in Vaspy, which are `RECORD_AUDIO` (to record the activation voice of the user), `INTERNET` (to dynamically fetch attacking commands from the Firebase server and interact with trained online model), and `READ_PHONE_STATE` (to monitor incoming/outgoing call status). Vaspy is disguised as a popular microphone-controlled game, so that it can legitimately request the `RECORD_AUDIO` permission without being suspected by the user. When a victim user plays the game, the player is required to blow or scream to the microphone to raise a rocket. The higher volume the microphone receives, the faster the rocket flies. (The snapshot of the game can be found in Figure 3). The game is very deceivable to teenagers or kids. In fact, the spyware can be delivered in other forms such as a malicious audio recorder. `READ_PHONE_STATE` and `INTERNET` permissions are very commonly requested by various Android games. There are 46 of the top 100 games on Google Play that requests the `READ_PHONE_STATE` permission, while all of the top 10 games request the `INTERNET` permission.

V. EVALUATION

In this section, we evaluate the performance of our prototype spyware in terms of the attack success rate. The attack capabilities on the VAs from various vendors are also investigated. We evaluate the proposed attack on three VAs on four Android smartphones, including Google Assistant on Google Pixel 2 and Samsung Galaxy S9, Xiao Yi on Huawei Mate 8, and Xiao Ai on Xiaomi Mi 8. In addition, to examine its stealthiness, we evaluate the system overhead, and teste Vaspy against anti-virus tools/platforms.

1: Collecting time, 2: Activity, 3: Avg. noise level, 4: Avg. light level (phone in pocket), 5: Avg. light level (phone on hand)

FIGURE 4. **Overview of the real-world scenarios.**

A. EVALUATION OF THE ACTIVATION VOICE MANIPULATION MODULE

To evaluate the effectiveness of the AVM module in the spyware, we built a simple corpus, which includes 60 positive sentences (*i.e.*, sentences containing the required activation words) and 40 negative sentences (*i.e.*, sentences containing no required activation words). The location of the required activation words in the positive sentences varies. For instance, the activation words can appear at the beginning, in the middle or at the end. We recruited 10 participants (5 male, 5 female) to read these sentences. We provided Google Pixel 2 and Samsung Galaxy 9 to record the audio. In addition, we asked these 10 participants to speech each sentence in various speed to improve diversity.

The recorded audio clips are then input to the AVM module one by one. If the module correctly detects and extract the activation word from a positive sentence, it will be marked as a success; otherwise, will be marked as a failure. The model should ignore all negative sentences, therefore, any activation word extracted from a negative sentence will be counted as a failure. Eventually, we achieve 95% (57/60) accuracy in positive sentences and 100% (40/40) accuracy in negative sentences

B. EVALUATION OF THE ATTACKING ENVIRONMENT SENSING MODULE

Smartphones are taken to various real-world scenarios for data collection. These scenarios include moving or stationary states, noisy or quiet environment. The data are collected in different time period of a day. The example scenarios are shown in Figure 4. In each of the scenarios, we collected the data when the phone was held on hands as well as put in the pocket.

Each smartphone is carried by a participant for data collection. An audio piece of synthesised activation voice is stored in each smartphone. These activation voices are tested in advance to make sure that they can successfully activate the voices assistant on the smartphones. In every two minutes, the activation voice followed by one random attacking voice command (*e.g.*, "Send 'subscribe' to 1234567") is played via smartphone's built-in speaker. If the participant does not notice the voice command, and the command is successfully executed, we label this attack as success. The data we collected for training includes the readings from smartphone on-board sensors (*i.e.*, microphone, accelerometer, and ambient light sensor) and attack results (as label set). We collect 30 group data in each scenario, 10 of them are under the condition of holding the phone on hands, the rest are under conditions of putting the phone in a pocket. Table 1 shows all the attack result of training data over eight example scenarios. It reveals that attacks are almost failed when participants hold the phone on hands. The highest success rate is achieved when participants are in a noisy environment with the phone in their pockets.

We process the collected data to further train the AES module. Accelerometer data is collected every 20 *ms*, while noise and light data are collected every 200 *ms*, as they are more stable in a short period of time. Noise and light data are re-sampled to the frequency of 50 *Hz* by following the Nearest Neighbour Interpolation principle [41], and merged with one-hot encoded *movement intensity* features to built the training matrices. The features of *environment variables* are used for the purpose of providing more specific details on the

TABLE 1. Attacking success rate of training data in each scenario.

Scenarios		Success Rate
(a) Quiet road	On hands	0 / 10
	In pocket	0 /20
(b) Highway	On hands	2 / 10
	In pocket	20 /20
(c) Market & Uni	On hands	1 / 10
	In pocket	11 /20
(d) Tram	On hands	1 / 10
	In pocket	17 /20
(e) Car	On hands	0 / 10
	In pocket	0 /20
(f) Restaurant	On hands	0 / 10
	In pocket	20 /20
(g) Quiet road (night)	On hands	0 / 10
	In pocket	0 /20
(h) Highway (night)	On hands	0 / 10
	In pocket	3 /20

TABLE 2. Average accuracy performance.

Invasion	Precision	Recall	f1-score
Unsuccessful	0.96	0.95	0.95
Successful	0.97	0.98	0.98
Avg	0.97	0.97	0.97

uncertain environmental factors, such as noise level and light intensity, which can also affect the decision on whether to launch a stealthy attack.

The collected raw signals usually contain noise generated by different sources, such as sensor miscalibration, sensor errors, noisy environments, *etc*. These noisy signals adversely affect the signal segmentation and feature extraction, and further significantly hamper the activity prediction. In our study, we use a fourth-order Butterworth low-pass filter [42] for noise removal. Except gathering the data from on-board sensors, we also collect smartphone usage status, such as the lock screen on/off status and the Bluetooth/headphone connection status by invoking corresponding APIs. These status indicate whether the smartphone is in use. Environment sensing will be triggered only when the smartphone is not in use.

We train a Random Forest with collected data, and evaluated the model based on Precision, Recall, and F1 Score. The results of 20-fold cross validation is presented in Table.2. It shows the proposed model is well-trained.

C. EVALUATION OF REAL WORLD ATTACK

We further evaluate the effectiveness of the attack in real-world scenarios in different times of a day. Ten participants are recruited to carry one of the aforementioned smartphones to various real-world scenarios. Smartphone sensors collect environment data and feed it to the online trained machine learning model every two minutes. Then, a probability of whether to launch an attack is obtained. An attack will be triggered if the probability exceeds a threshold (*e.g.*, 80% in our experiment setting). We set up a restriction that in every two minutes, there will be at most one attack triggered. Figure 5 reports the sensors' readings and the output attacking probabilities in some typical scenarios, where "True" in the attack results indicates that the attack is triggered but not heard by the participant, while "False" represents that the attack is triggered and heard by the participants. "N/A" means that no attack is triggered in the time slot, so that it is excluded when calculating the success rate. We can see from Figure 5 that the spyware achieves 100% success rate in real world attacks. For example, in5 (a), the participant is having lunch in a food court and the environment is noisy and crowded. AES module launched 11 attacks in 30 minutes, and all attacks are successful. In Figure 5 (b), participants sit in an office. The sensors' data are stable and the environment is quiet. Under these conditions, there is no attack launched. Figure 5 (c) and Figure 5 (d) are in similar environments, the only difference is weather the participant holds the phone or not. The result shows no attack launched when the participant holds the phone on hand, and 4 successful attacks launched when the phone is put in a pocket.

D. CAPABILITY OF ATTACK

After activating the VAs, the attackers may further acquire victim's private information [43], or conduct malicious activities on the infected smartphones, through remotely executing specific attacking commands.

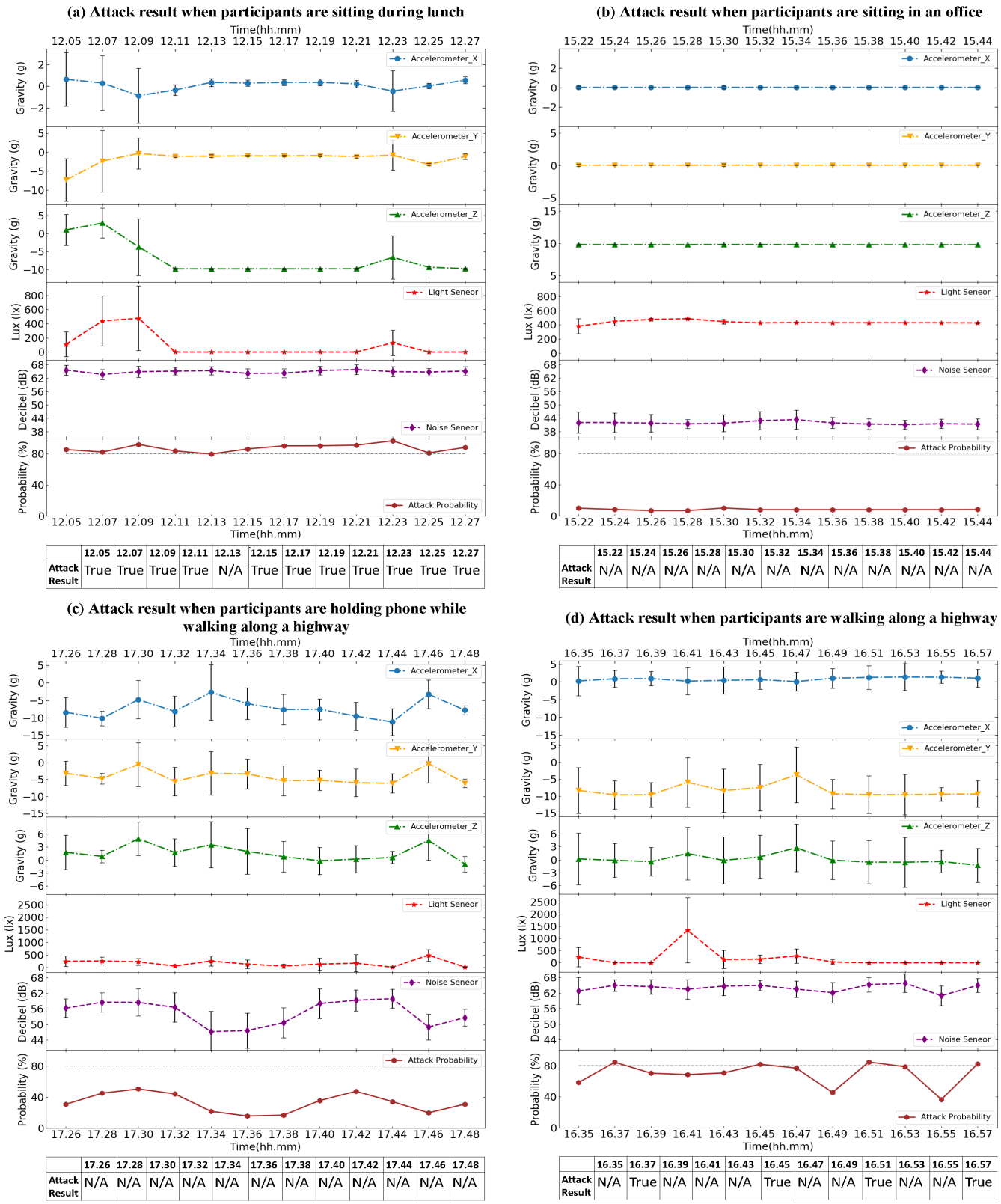
In Table. 3, we list and compare the potential attacks that can be launched on different VAs in victim smartphones, namely Google Assistant on Pixel 2, Xiao Yi on Huawei Mate 8, and Xiao Ai on Xiaomi Mi 8. We also listed the permissions required if the corresponding information are queried in an app. However, none of these permissions are required in the proposed attack, since VAs are naturally gained privilege to access such information.

While private information such as location, calendar, IP address *etc.*, can be queried locally, most of them cannot be sent out as text, with one exception that Google Assistant can send user's current location via SMS to arbitrary number. However, this does not necessarily mean that attackers cannot access these information remotely. Actually, an attacker can manipulate VA to start a phone call to him, and then query the private information during the phone call. The audio response from the VA can then be heard by the attacker.

The malicious activities such as making phone calls to premium numbers, sending SMS, browsing malicious websites and so on, can be performed on all the VAs that we tested without requesting for any permissions.

Many new features (*e.g.,* smart-home devices) are controlled by VAs now. Once user set up smart devices on VAs, a command such as "unlock the front door" can control the smart lock on a door remotely. Such malicious activities may bring higher risk compare to privacy leak on smartphones [44]. Theoretically, any IoT devices controlled by VAs can be hacked by Vaspy [45], [46]. Users always talk about how convenient a VA can be, but still did not realise that VA is becoming a powerful hacking tool as well.

Attacking commands are stored either locally in the spyware, or remotely in Firebase as text scripts. Commands



	12.05	12.07	12.09	12.11	12.13	12.15	12.17	12.19	12.21	12.23	12.25	12.27
Attack Result	True	True	True	True	N/A	True	True	True	True	True	True	True

	15.22	15.24	15.26	15.28	15.30	15.32	15.34	15.36	15.38	15.40	15.42	15.44
Attack Result	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

	17.26	17.28	17.30	17.32	17.34	17.36	17.38	17.40	17.42	17.44	17.46	17.48
Attack Result	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

	16.35	16.37	16.39	16.41	16.43	16.45	16.47	16.49	16.51	16.53	16.55	16.57
Attack Result	N/A	True	N/A	N/A	N/A	True	N/A	N/A	True	N/A	N/A	True

FIGURE 5. Evaluation Result of the Attacking Environment Sensing. The average value and the standard deviation of sensor's data in every two-minute time period is showed in the figure. The attacking results are listed below, where "True" indicates a successful attack without users' awareness,"N/A" means no attack was triggered.

stored online will be downloaded to the local smartphone once it determines to launch an attack. The text commands are converted to speech using Android built-in Text-to-speech API, and played to control the VA via the built-in speaker.

TABLE 3. Post attack commands against VAs.

Category	Attack type	Permission(s) bypassed	Attack result against VAs		
			Google	Huawei	Xiaomi
Privacy leak	Query location	ACCESS_COARSE_LOCATION	√	√	√
	Share location	READ_CONTACTS, SEND_SMS, WRITE_SMS, ACCESS_COARSE_LOCATION,	√	×	×
	Query calendar	READ_CALENDAR	√	√	√
	Share calendar	READ_CALENDAR,READ_CONTACTS, SEND_SMS,WRITE_SMS	×	×	×
	Query ip address	ACCESS_COARSE_LOCATION, INTERNET	√	√	√
	Share ip address	ACCESS_COARSE_LOCATION, INTERNET, READ_CONTACTS, SEND_SMS, WRITE_SMS	×	×	×
Malicious activity	Phone Call	READ_CONTACTS, CALL_PHONE	√	√	√
	Send SMS	READ_CONTACTS, SEND_SMS, WRITE_SMS	√	√	√
	Send email	READ_CONTACTS, INTERNET	√	√	√
	Browse website	INTERNET	√	√	√
	Bluetooth control	BLUETOOTH	√	√	√
	Video call	INTERNET,CAMERA	√	√	√
	Play music	INTERNET	√	√	√
	Control lighting	INTERNET	√	√	√
	Control camera	HARDWARE_CAMERA, ACCESS_FINE_LOCATION, WRITE_EXTERNAL_STORAGE	√	√	√

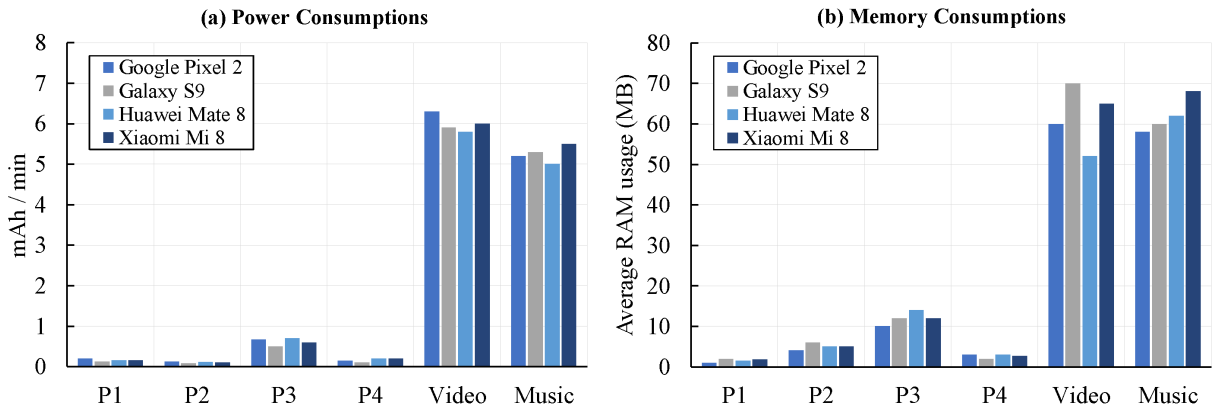


FIGURE 6. Power and memory consumption of four phases: P1(Phone call state monitoring), P2(Recording and synthesising activation command), P3(Environment monitoring), and P4(Attacking via the speaker).

Given the fact that VAs can be easily controlled by attackers to perform malicious activities as well as acquiring private information, we suggest that the vendors should rethink the privilege assigned to VAs.

E. RUNTIME COST ANALYSIS

We evaluate and analyse the runtime cost of the spyware because high runtime cost (*e.g.*, CPU, Memory) will reduce the stealthiness of the attack. We install the prototype spyware on Google Pixel 2, Huawei mate 8, Xiaomi Mi 8 and Samsung Galaxy S9. Since the spyware launches the attack in the four distinctive phases below, we evaluate each phase individually: P1(Phone call state monitoring), P2(Recording and synthesising activation command), P3(Environment monitoring), and P4(Attacking via a speaker).

1) POWER CONSUMPTION ANALYSIS

Figure 6 (a) reports the power consumption per minute for four attacking phases. We also compare the power consumption with playing 1080P video and music. The results show that in P1, P2, and P4, the power consumption per minutes on all Android phones are very low. P3 has the highest power consumption, which is approximately 0.8 *mAh* per minute. It is still negligible when compared with the scenarios such as playing video or listening to music, which consumes 6.1 *mAh* and 5.1 *mAh* per minute, respectively. We further reduce the frequency of collecting data from sensors in P3 from 50 *Hz* to 10 *Hz*. The power consumption decreases to 0.5 *mAh*, without affecting the success rate of the attack. The results suggest that the spyware consumes too little power to be noticed by the user.

2) MEMORY AND CPU ANALYSIS

Figure 6 (b) shows the average RAM usage in the four processes. The average RAM usage in P1, P2, and P3 is less than 5 *MB*. The P3 uses the highest memory (approximately 10 *MB*) because of sensor utilisation. Compared to the scenarios like playing video or listening to music, which consumes approximately 60 *MB* to 70 *MB*, the memory cost of our prototype spyware can hardly affect the performance

of the hosting smartphone systems. Therefore, it is hard to be noticed by the user. We also evaluate the CPU cost. It is found that only P3 requires CPU, which consumes around 7% of the total capacity.

3) FILE SIZE

In P2 and P3, the recorded voice pieces and sensors' data will be stored until they are uploaded to the server. There is no need to store file in P1 and P4. Therefore, during the whole attacking process, only two categories of files are stored then uploaded to server: an audio file (*.wav) to store the synthesised activation voice, and three text files (*.txt) to record the sensors data. The average size of voice, acceleration, light, noise files are 180.9 KB, 91.7 KB, 4.4 KB, 5.4 KB respectively.

F. RESISTANCE TO ANTI-VIRUS TOOLS

We test the spyware against industrial anti-malware tools as well as academic malware detection solutions. Android malware detection approaches can be categorised into static tools and dynamic platforms, according to whether the candidate app needs to be executed or not [47]. Static approaches are based on analysing static features of the application, such as the component of the app, the permissions requested by the application, and the code itself. Dynamic approaches execute the application in a protected environment, provide all the emulated resources it needs, and observe its malicious activities.

For industrial anti-virus products, we test the spyware on VirusTotal, as well as the top ten most popular anti-virus tools on Google Play, such as Norton Security and Antivirus, Kaspersky Mobile Antivirus, McAfee Mobile Security, and so on. None of them reported our spyware as malicious app. We also submit the spyware to Google Play store, where submitted apps are tested against their dynamic test platform Google Bouncer. The spyware successfully passes the detection of Google Bouncer. Note that we took down the spyware from the Google Play immediately after it passed the test.

We also test the spyware with three typical learning-based detectors in academia, which rely on syntactic features (e.g., requested permissions, presence of specific API calls, etc.), as well as semantic features (e.g., sequence of API calls) extracted from Android application packcage (APK), namely Drebin [16], DroidAPIMiner [17], and MaMaDroid [18]. We trained all the detectors with 5,000 most recently discovered malware samples and 5,000 benign apps that we collected from Virusshare [4] and Google Play store between August and October 2018, respectively. Our spyware is labeled as a benign app by all three detectors. The results show the resistance of the proposed attacking method to both industrial and academic malware detection tools.

[4]https://virusshare.com/

VI. DISCUSSION

A. IMPORTANCE OF ENVIRONMENT SENSING

Once the attack is noticed by users, they may look into this issue, and there would not be a second chance to launch the attack. Existing work [1], [3] gathers some state values of the Android phone from sensors on the device (*e.g.,* light sensor and accelerometer) and system attributes (*e.g,* current system time and screen on or off status). They usually trigger the attack only if those state values satisfy a given profile. For example, in [3], attacks are only launched at night, when the phone's screen is off and the phone is placed on a horizontal table in a dark room. As different users may have different habits, relying on such pre-set conditions may result in an attack that is never triggered or easily noticed.

B. SMARTER VASPY

The Vaspy proposed in Section IV-A intends to make the attack stealthier. The proposed attack can be further improved to be more energy efficient. Once the activation voice is synthesised, the environment detector keeps collecting data from sensors. As we discussed in Section V, though the power consumption is very low, it still has a chance to be noticed by the user if it keeps running for a long time. We observed that in the situations where the Vaspy decides to launch the attack, the noise levels are higher than a certain threshold in most cases. As a result, the noise level can be a preliminary indicator for the detection algorithm. On the standby mode, only microphone is activated to collect noise data. Once the noise level exceeds a certain threshold, the other two sensors then start to collect data. With this approch, we have tested that the power consumption can be further reduced from 0.8 mAh to 0.4 mAh, which makes our prototype spyware even harder to be detected by the user.

C. ATTACKING VOLUME

The volume of the activation and attacking voice is also an important factor that affects the success of the attack. The volume should be low enough to avoid being heard by the user, and high enough to make sure the attacking voice command is received by the Android phone. The optimal volume varies depending on the noise level of the surroundings as well as the real-world scenario that the phone is in. Based on this finding, Vaspy is designed to launch the attack and use the optimal sound volume according to the ambient environment of the Android phone user.

D. ESSENTIAL FACTORS FOR THE SUCCESSFUL ATTACK

We demonstrate three core essential factors for the successful attack: 1) success in delivery, 2) success in avoiding being detected by anti-malware products, and 3) success in not being noticed by users.

1) The prevalent abuse of high risk permissions in Android applications makes users insensitive to granting these permissions. For example, with the rise of popularity of location-based applications and Augmented

Reality (AR) games, Android users are becoming less cautious when granting the permission to access the location and the camera, both of which are high risk permissions that may leak users' private information. In our proof-of-concept attack, we disguise *Vaspy* as a microphone controlled game to trick the user for granting the permission to access the microphone.

2) Currently, commercial anti-malware tools are based on known features of malware, such as signatures and sensitive operations. In Vaspy, there is no relevant data in existing signature library of anti-malware tools. All the sensitive operations in the attack, such as sending emails and making phone calls, are executed through the VA. It has privilege to access sensitive data but it is not monitored by the anti-malware products.

3) In the proposed attack, we developed a stealthy attacking module inside Vaspy, which monitors the environment and looks for suitable time to launch the attack. It also adjusts the volume of the voice commands, to ensure that the voice commands can be captured and recognised by the Android phone, but it cannot be heard by users.

E. DEFENCE APPROACHES FOR VASPY

In this section, we demonstrate three possible defence approaches for Vaspy: 1) identifying the source of the voice commands; 2) continuous authentication for VAs; 3) distinguishing human voice from machine-based voice.

1) Identifying the source of the voice commands. In the proposed attack scenario, the voice commands are played via a speaker on a smartphone. New techniques [48] are able to locate the source of the sound, which can then determine whether the sound comes from the built-in speaker. The VA vendors can disable our attack by setting the VA to disregard any voice commands from the built-in speaker on its hosting smartphone.

2) Continuous authentication for VAs. Feng *et al.* [49] propose a scheme that collects the body-surface vibrations of the user and matche with the speech signal received from a microphone. The VA only executes the commands that originate from the owner's voice. While it may successfully defend our attack, it also brings some inconvenience to the user. For example, users cannot activate the VA when they do not hold the smartphone. Actually, users tend to interact with VA when they are not able to touch the screen, such as the time when they are driving.

3) Distinguish human voice from machine-based voice. Chen *et al.* [50] explored the difference between a human voice and machine-based voice based on the magnetic field emitted from loudspeakers. It is able to detect machine-based voice impersonation attacks. However, there will be high false positive rate when there are other devices around, which generates magnetic signals.

F. LESSONS FROM THIS WORK

This can be recognised as a vulnerability in the current VAs. Once the VAs are activated, they are able to change smartphone settings, and do malicious activities that require high level permission, such as sending SMS/emails and making phone calls. Due to the privilege it has to access system resources and private information. VAs can then be a stepping stone for the attackers to hack into the Android phones. More secure mechanisms will be implemented to improve the security of VAs, from either the research community or the VA vendors.

VII. CONCLUSION

In this paper, we propose a smart and stealthy attack Vaspy targetting VAs on Android phones. With the new attack, an attacker can forge voice commands to activate the VA and launch a number of attacks, including leaking private information, sending forged Message or emails, and calling arbitrary numbers. An Attacking Environment Sensing module is built inside the Vaspy to choose an optimal attacking time and voice volume making the attack unnoticed by the users. We build a prototype spyware for Vaspy and evaluate the spyware with participants across various VAs on different Android phones. We demonstrate that Vaspy is able to launch attacks without being noticed by users. Moreover, our spyware cannot be detected by the state-of-art anti-malware tools from both industry and academia. We also propose a few potential solutions to detect our attack. This research work may inspire the researchers for Android phones to strengthen the security of VAs in general.

REFERENCES

[1] E. Alepis and C. Patsakis, "Monkey says, monkey does: Security and privacy on voice assistants," *IEEE Access*, vol. 5, pp. 17841–17851, 2017.

[2] R. Knote, A. Janson, L. Eigenbrod, and M. Söllner, "The what and how of smart personal assistants: Principles and application domains for IS research," Multikonferenz Wirtschaftsinformatik, Lüneburg, Germany, 2018.

[3] W. Diao, X. Liu, Z. Zhou, and K. Zhang, "Your voice assistant is mine: How to abuse speakers to steal information and control your phone," in *Proc. 4th ACM Workshop Secur. Privacy Smartphones Mobile Devices (SPSM)*, Nov. 2014, pp. 63–74.

[4] J. Aron, "How innovative is Apple's new voice assistant, Siri," *NewScientist*, vol. 212, no. 2836, p. 24, Oct. 2011.

[5] J. Kiseleva, K. Williams, A. H. Awadallah, A. C. Crook, A. C. Crook, and T. Anastasakos, "Understanding user satisfaction with intelligent assistants," in *Proc. ACM Conf. Hum. Inf. Interact. Retr. (CHIIR)*, Mar. 2016, pp. 121–130.

[6] S. Yu, S. Guo, and I. Stojmenovic, "Fool me if you can: Mimicking attacks and anti-attacks in cyberspace," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 139–151, Jan. 2013.

[7] (2018). *Control Google Home by Voice*. Accessed: Sep. 29, 2018. [Online]. Available: https://support.google.com/googlehome/answer/7207759?hl=en-AU&ref_topic=7196346

[8] (2018). *Now the Google Assistant Can Take Care of Your Ious*. Accessed: Sep. 29, 2018. [Online]. Available: https://www.blog.google/products/assistant/now-google-assistant-can-take-care-your-ious

[9] Google. (2018). *What Can Your Google Assistant do*. Accessed: Sep. 29, 2018. [Online]. Available: https://assistant.google.com/explore?hl=en-AU

[10] S. Yu, G. Wang, and W. Zhou, "Modeling malicious activities in cyber space," *IEEE Netw.*, vol. 29, no. 6, pp. 83–87, Nov./Dec. 2015.

[11] N. Carlini, P. Mishra, T. Vaidya, Y. Zhang, M. Sherr, C. Shields, D. Wagner, and W. Zhou, "Hidden voice commands," in *Proc. 25th USENIX Secur. Symp. (USENIX Secur.)*, 2016, pp. 513–530.

[12] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu, "DolphinAttack: Inaudible Voice Commands," in *Proc. 2017 ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Nov. 2017, pp. 103–117.

[13] N. Xu, F. Zhang, Y. Luo, W. Jia, D. Xuan, and J. Teng, "Stealthy video capturer: A new video-based spyware in 3G smartphones," in *Proc. 2nd ACM Conf. Wireless Netw. Secur.*, Mar. 2009, pp. 69–78.

[14] S. Arik, J. Chen, K. Peng, W. Ping, and Y. Zhou, "Neural voice cloning with a few samples," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 10019–10029.

[15] *Virus Total*. Accessed: Sep. 29, 2018. [Online]. Available: https://www.virustotal.com

[16] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "Drebin: Efficient and explainable detection of Android malware in your pocket," in *Proc. NDSS*, Feb. 2014, pp. 23–26.

[17] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust Malware detection in android," in *Proc. Int. Conf. Secur. Privacy Commun. Syst.* Cham, Switzerland: Springer, 2013, pp. 86–103.

[18] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "MaMaDroid: Detecting Android malware by building Markov chains of behavioral models," 2016, *arXiv:1612.04433*. [Online]. Available: https://arxiv.org/abs/1612.04433

[19] A. Muzet, "Environmental noise, sleep and health," *Sleep Med. Rev.*, vol. 11, no. 2, pp. 135–142, Apr. 2007.

[20] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. Cham, Switzerland: Springer, 2013, pp. 387–402.

[21] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren, "Android HIV: A study of repackaging malware for evading machine-learning detection," *IEEE Trans. Inf. Forensics Security*, vol. 15, no. 1, pp. 987–1001, Jul. 2019.

[22] D. Kumar, R. Paccagnella, P. Murley, E. Hennenfent, J. Mason, A. Bates, and M. Bailey, "Skill squatting attacks on amazon alexa," in *Proc. 27th USENIX Secur. Symp. (USENIX Secur.)*, 2018, pp. 33–47.

[23] L. Schönherr, K. Kohls, S. Zeiler, T. Holz, and D. Kolossa, "Adversarial attacks against automatic speech recognition systems via psychoacoustic hiding," 2018, *arXiv:1808.05665*. [Online]. Available: https://arxiv.org/abs/1808.05665

[24] X. Yuan, Y. Chen, Y. Zhao, Y. Long, X. Liu, K. Chen, H. Huang, X. Wang, and C. A. Gunter, "Commandersong: A systematic approach for practical adversarial voice recognition," in *Proc. 27th USENIX Secur. Symp. (USENIX Secur.)*, 2018, pp. 49–64.

[25] L. C. De Silva, "Multi-sensor based human activity detection for smart homes," in *Proc. 3rd Int. Universal Commun. Symp. (IUCS)*, Dec. 2009, pp. 223–229.

[26] J. Wiese, T. S. Saponas, and A. B. Brush, "Phoneprioception: Enabling mobile phones to infer where they are kept," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst. (CHI)*, May 2013, pp. 2157–2166.

[27] X. Liu, Z. Zhou, W. Diao, Z. Li, and K. Zhang, "When good becomes evil: Keystroke inference with smartwatch," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Oct. 2015, pp. 1273–1285.

[28] P. Marquardt, A. Verma, H. Carter, and P. Traynor, "(sp)iPhone: Decoding vibrations from nearby keyboards using mobile phone accelerometers," in *Proc. 18th ACM Conf. Comput. Commun. Secur.*, Oct. 2011, pp. 551–562.

[29] J. Ho and S. S. Intille, "Using context-aware computing to reduce the perceived burden of interruptions from mobile devices," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, Apr. 2005, pp. 909–918.

[30] A. Anjum and M. U. Ilyas, "Activity recognition using smartphone sensors," in *Proc. IEEE 10th Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2013, pp. 914–919.

[31] M. Shoaib, S. Bosch, O. D. Incel, H. Scholten, and P. J. Havinga, "Fusion of smartphone motion sensors for physical activity recognition," *Sensors*, vol. 14, no. 6, pp. 10146–10176, Jun. 2014.

[32] A. Krause, A. Smailagic, and D. P. Siewiorek, "Context-aware mobile computing: Learning context- dependent personal preferences from a wearable sensor array," *IEEE Trans. Mobile Comput.*, vol. 5, no. 2, pp. 113–127, 2005.

[33] *Firebase*. (2019). [Online]. Available: https://firebase.google.com

[34] J. Binder, S. D. Post, O. Tackin, and T. R. Gruber, "Voice trigger for a digital assistant," U.S. Patent 2 014 022 436 A1, Aug. 7, 2014.

[35] A. Gibiansky, S. Arik, G. Diamos, J. Miller, K. Peng, W. Ping, J. Raiman, and Y. Zhou, "Deep voice 2: Multi-speaker neural text-to-speech," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 2962–2970.

[36] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 6645–6649.

[37] *Ibm Watson*. Accessed: Sep. 28, 2018. [Online]. Available: https://www.ibm.com/watson/

[38] A. Ganapathiraju, J. Hamaker, J. Picone, M. Ordowski, and G. R. Doddington, "Syllable-based large vocabulary continuous speech recognition," *IEEE Trans. Speech Audio Process.*, vol. 9, no. 4, pp. 358–366, May 2001.

[39] (2018). *Android 9 Changes*. Accessed: Apr. 1, 2019. [Online]. Available: https://developer.android.com/about/versions/pie/android-9.0-changes-all

[40] K. Cho, B. Van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," 2014. *arXiv:1409.1259*. [Online]. Available: https://arxiv.org/abs/1409.1259

[41] *Nearest Neighbor Interpolation*. (2019). [Online]. Available: https://en.wikipedia.org/wiki/Nearest-neighbor_interpolation

[42] D. Ahn, J. S. Park, C. S. Kim, J. Kim, Y. Qian, and T. Itoh, "A design of the low-pass filter using the novel microstrip defected ground structure," *IEEE Trans. Microw. Theory Techn.*, vol. 49, no. 1, pp. 86–93, Jan. 2001.

[43] T. Wu, S. Wen, Y. Xiang, and W. Zhou, "Twitter spam detection: Survey of new approaches and comparative study," *Comput. Secur.*, vol. 76, pp. 265–284, Jul. 2018.

[44] J. Jiang, S. Wen, S. Yu, Y. Xiang, and W. Zhou, "Identifying propagation sources in networks: State-of-the-art and comparative studies," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 465–481, 1st Quart., 2017.

[45] Y. Qu, S. Yu, W. Zhou, S. Peng, G. Wang, and K. Xiao, "Privacy of things: Emerging challenges and opportunities in wireless Internet of Things," *IEEE Wireless Commun.*, vol. 25, no. 6, pp. 91–97, Dec. 2018.

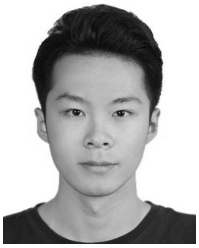
[46] I. Stojmenovic, S. Wen, X. Huang, and H. Luan, "An overview of fog computing and its security issues," *Concurrency Comput., Pract. Exper.*, vol. 28, no. 10, pp. 2991–3005, 2016.

[47] S. Peng, S. Yu, and A. Yang, "Smartphone malware and its propagation modeling: A survey," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 2, pp. 925–941, 2nd Quart., 2014.

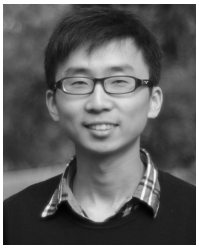
[48] R. Liu, C. Cornelius, R. Rawassizadeh, R. Peterson, and D. Kotz, "Poster: Vocal resonance as a passive biometric," in *Proc. 15th Annu. Int. Conf. Mobile Syst., Appl., Services (MobiSys)*, Jun. 2017, p. 160.

[49] H. Feng, K. Fawaz, and K. G. Shin, "Continuous authentication for voice assistants," in *Proc. 23rd Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, Oct. 2017, pp. 343–355.

[50] S. Chen, K. Ren, S. Piao, C. Wang, Q. Wang, J. Weng, L. Su, and A. Mohaisen, "You can hear but you cannot steal: Defending against voice impersonation attacks on smartphones," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 183–195.



RONGJUNCHEN ZHANG received the B.Sc. degree in software engineering from Southwest University, China, and Deakin University, Australia, in 2018, and the B.Sc. degree (Hons.) from the Swinburne University of Technology, in 2018, where he is currently pursuing the Ph.D. degree. His main research interests include natural language processing, reinforcement learning, and security of chatbot.



XIAO CHEN received the B.Eng. degree from Hangzhou Dianzi University, China, in 2010, the M.IT. degree from the University of Melbourne, Australia, in 2012, the M.Sc. degree from Deakin University, Australia, in 2014, and the Ph.D. degree from the Swinburne University of Technology, Australia, in 2019. His research interests include mobile security, network traffic analysis, and adversarial machine learning.

SHENG WEN received the Ph.D. degree from Deakin University, Australia, in October 2014. He has been received over three million Australia Dollars funding from both academia and industries, since 2014. He is currently a Senior Lecturer with the Swinburne University of Technology and a Visiting Scholar with the Peng Cheng Laboratory, Cyberspace Security Research Center, China. He is also leading a Mediumsize Research Team in cybersecurity area. He has published more than 50 high-quality articles in the last six years in the fields of information security, epidemic modeling and source identification. His research interests include social network analysis and system security. His representative research outcomes have been mainly published on top journals, such as the IEEE Transactions on Computers (TC), the IEEE Transactions on Parallel and Distributed Systems (TPDS),the IEEE Transactions on Dependable and Secure Computing (TDSC), the IEEE Transactions on Information Forensics and Security (TIFS), and the IEEE Communication Survey and Tutorials (CST).

XI ZHENG received the bachelor's degree in computer information system from FuDan; Chief Solution Architect for Menulog Australia, the master's degree in computer and information science from UNSW, and the Ph.D. degree in software engineering from UT Austin. He is currently the Deputy Director of Software engineering, Global Engagement, and an Assistant Professor in software engineering with Macquarie University. Specialized in service computing, the IoT security, and reliability analysis. He has published more than 50 high quality publications in top journals and conferences, such as PerCOM, ICSE, the IEEE Transactions on Cybernetics, the IEEE Transactions on Industrial Informatics, the IEEE Transactions on Vehicular Technology, the IEEE IoT Journal, and the *ACM Transactions on Embedded Computing Systems*. He was a recipient of the Best Paper in Australian distributed computing and doctoral conference, in 2017, and the Deakin Research Outstanding Award, in 2016. His article is recognized as a top 20 most read article, from 2017 to 2018, in *Concurrency and Computation: Practice and Experience*. He is the Guest Editor and a PC Member for top journals and conferences, such as the IEEE Transactions on Industry Informatics, *Future Generation Computer Systems*, and PerCOM. He is also the WiP Chair for PerCOM 2020 and the Track Chair for CloudCOM 2019. He is the Publication Chair for ACSW 2019 and a Reviewer for many transactions, journals, and CCF A/CORE A* conferences.

YONG DING was a Research Fellow of computer science with the City University of Hong Kong, from April 2008 to September 2009. He is currently a Professor with the School of Computer Science and Information Security, Guilin University of Electronic Technology, China, and an Adjunct Professor with the Cyberspace Security Research Center, Peng Cheng Laboratory, China. His research interests include cryptography and information security.