Monkey Says, Monkey Does: Security and Privacy on Voice Assistants

Efthimios Alepis, Constantinos Patsakis, *Member, IEEE.*

***Abstract*—The introduction of smart mobile devices has radically redesigned user interaction, as these devices are equipped with numerous sensors, making applications context-aware. To further improve user experience, most mobile operating systems and service providers are gradually shipping smart devices with voice controlled intelligent personal assistants, reaching a new level of human and technology convergence. While these systems facilitate user interaction, it has been recently shown that there is a potential risk regarding devices which have such functionality. Our independent research indicates that this threat is not merely potential, but very real and more dangerous than initially perceived, as it is augmented by the inherent mechanisms of the underlying operating systems, the increasing capabilities of these assistants, and the proximity with other devices in the IoT era. In this work, we discuss and demonstrate how these attacks can be launched, analysing their impact in real world scenarios.**

***Index Terms*—Security, Voice recognition, mobile devices, Android permissions, Voice assistants**

I. Introduction

We are currently witnessing the continuous growth of smart devices and their accompanied services in both numbers and usage. The main reasons for this radical shift can be attributed to the incorporation of numerous sensors in these devices, as well as to the fact that users can easily install applications through a number of application marketplaces. Embedded sensors, such as cameras, accelerometers, or even GPS receivers, enable developers to make their applications more context-aware, radically improving user experience. While these improvements are significant and welcomed by the users, their arbitrary usage can become quite dangerous, as sensitive information such as real-time video or location can be leaked. To counter such security and privacy issues, mobile operating systems have introduced permission models that inform users of the privacy invasive resources that each application needs to access. This way, users can assess the risk they are exposed to and determine whether they agree to take it. In recent versions of most popular mobile operating systems, users may even grant these privileges on runtime and/or revoke them whenever deemed necessary.

To further improve user experience, virtually all mobile operating systems, as well as service providers, have gradually introduced voice assistants in their platforms, also known as "Intelligent Personal Assistants". In fact, these software entities have not only drawn the attention of major Operating System vendors, but also other software giants such as Samsung, Facebook and Amazon. Voice assistants are replacing traditional human-computer interaction, redefining the way we access the Web, data and apps. This facilitates users as they are not required to type commands or have any kind of physical interaction with their devices, realising the concept of ubiquitous computing, and eventually providing a more human-like experience. This seamless interaction with the smart device provides more user satisfaction [1], [2].

Generally, these systems operate silently in the background, waiting for specific keywords/commands, or more specifically "hot-words", from the users. While all of these systems can respond to explicit voice commands such as "Call Bob", most of them are able to understand, and consequently execute, more complex user requests such as "Find the closest gas station", or even start a dialog with a user in order to accomplish a complex task. As a result, they have evolved in performing a broader set of actions, whether this is to adjust light luminosity in a smart house, or order goods from a market, as they are able to interact with other IoT devices and services. This is rather alarming, as sophisticated voice assistants are able to execute an increasing number of high priority commands, such as commands that involve changing OS system settings. Voice assistants are aiming in "assisting" or even "replacing" user interaction and thus require a large set of software permissions to accomplish their tasks, as illustrated in Appendix B. In addition, voice assistants most often belong either to the OS or to the device manufacturer, hence they have even more elevated privileges than typical apps. Furthermore, Voice Assistants' use is becoming so popular [3], [4] that a significant percentage of mobile search queries are performed by them [5], [6] and a growing number of devices and more importantly third party applications have emerged, that provide new ways to interact directly with them.

While voice assistants are becoming part of our daily lives and major tech giants are competing in acquiring the largest market share, these new technologies are accompanied with new risks for their users, introducing a new attack vector. In this work we examine some new attacks that can be launched from voice assistants, proving that we are not yet familiar enough with the new concepts they realise, and also the fact that they have not reached the necessary maturity. To this end, we study the most well-known voice assistants, focusing mostly on Android. More precisely, we focus on the five major voice assistants, namely Google Assistant (formerly known as Google Now), S Voice, Cortana (for both Android and Windows), Alexa and Siri and illustrate how they can be exploited from applications or nearby devices that do not request any permission from the user, evidently without the user's consent and/or actual knowledge about it. In this regard, the presented

E. Alepis and C. Patsakis are with the Department of Informatics, University of Piraeus, Greece. {talepis,kpatsak}@unipi.gr

attacks are clearly local privilege escalation attacks, but since they can be used to exploit other devices they are also remote code execution attacks. While our baseline is Android AOSP, which at the time of writing is Android Nougat, essentially we discuss the Samsung "flavour" of Nougat, as S Voice is installed only in Samsung devices.

Road map: The rest of this work is organised as follows. The next section, Section II provides an overview of the related work. Then, in Section III we detail some attack scenarios for various voice command systems, highlighting what an adversary could achieve. The applicability of our attacks is discussed in Section IV. Finally, the article concludes, discussing future work and probable countermeasures.

II. Related work

A. Attacks to Voice Assistants

One could argue that the predecessors of voice assistants can be traced in accessibility tools that are offered in several operating systems. Jang et al. [7] managed to trick these systems into performing unauthorised commands, using voice among other means. Recognising the risks that the users are exposed to, using voice assistants, Diao et al. [8] presented several attacks. One of the basic characteristics of these attacks was that a malicious app triggers the voice assistant to call the adversary, who then issues his/her commands via the headset. Later, Vaidya et al. [9] and Carlini et al. [10] further extended the aforementioned attacks, exploring the possibilities of hiding voice commands, *e.g.* in videos, illustrating that an adversary could hide voice commands in something that humans perceive as noise, but would still retain all the characteristics for a computer system to consider it a valid voice command, thus tricking voice assistants into executing it. These reported attacks could enable an adversary to leak sensitive user information, such as users' location, or visit web pages which contain malware, to gain unauthorised access to smart devices. More advanced attacks may even use an FM antenna to transmit radio waves in order to trick a device into converting them into voice signals, as long as a pair of headphones with a microphone are plugged into its jack [11].

While the aforementioned attacks are quite serious, they can be considered as static, as they lack actual interaction with the attacker. Moreover, they include a number of preconditions in order to be launched, which makes them have a limited impact and finally they do not investigate the IoT dimension of the attack vector, which in many cases makes the attacks stealthier and much more difficult to defend against them. In this research we attempt to unlock these attacks to their full potential, not only in terms of impact, but also stealthiness, as we present novel techniques to launch them in a more automated and responsive way. Additionally, we propose an attack methodology in order to create a chain of attacks, managing to successfully attack other devices in proximity as well.

The core difference of our approach is that our attacks are interactive, hence able to execute more advanced commands, or bypass security measures in an automated way. Furthermore, not only do we consider the case of local exploitation, which results to the obvious local privilege escalation attacks that previous researchers pointed out, but we also discuss extensions that could be added to achieve remote code execution attacks. Additionally, we detail how one could leak further sensitive information from users without the need to request dangerous permissions, such as using microphone input, like Diao et al. [8]. Finally, we illustrate how these attacks can be further extended, becoming far more severe, by utilising forced and implicit machine-to-machine communication.

B. Android permissions

The attacks which will be presented in this work are initiated from but are not necessary limited to malicious Android applications. Despite the fact that Android smartphones are by far the most widely used, the choice of Android was also made because of several inherent vulnerabilities in its workflow that can be exploited to execute possible attacks. The most profound one is that voice assistants can be launched remotely and without requesting any privileges from the user.

At this point it is worth highlighting two core differences in the two major mobile ecosystems, which are relevant to this work. Firstly, the use of intents, a mechanism in Android which allows applications to internally exchange information, facilitates developers in many ways, it simultaneously opens the door for various attacks, as malicious apps can utilize them to load their payload and exploit other apps. This mechanism does not exist in iOS, since URL Schemes seem to be the only way inside this OS to achieve inter-app communication. Secondly, contrary to Android, to succeed in launching Apple's voice assistant through an app, a rooted device is needed and certainly such an app cannot pass through the App Store's filters as the requested permission can only be granted by Apple.

It is also essential to discuss some details of the Android permission model since by exploiting it, we will execute our attacks and hide part of their traces. Up to recently, once a user wanted to install an app, Android would display a notification screen, informing him/her of the access permissions that he/she would have to grant. This was a one time offer, applying a "take it or leave it" policy, as users would have to accept the offer and grand the permissions to install it. Under the pressure of the custom modding communities and iOS who realised customisable permission policies, Google revised its permission model in Marshmallow, allowing users to apply fine-grained access policies. Thus, permissions are now granted after installation and applied during runtime, and most importantly, they can be revoked when deemed necessary.

The aforementioned permissions involve access to numerous smartphone embedded sensors. While for some of them their use seems quite straightforward, they can be used to infer a lot of other information, so gradually, many applications have started requesting more and more permissions. This way, they harvest user data and monetize them through data analytics. As a result, simple applications often require absurd permissions [12], gradually relaxing users' tolerance to requests for sensitive permissions, making users ignore them, without understanding the risks they can be exposed to in the end

[13], [14], [15]. The revised permissions to resources can be categorized according to the risk implied when granting them as shown in Table II and are the following.

Normal: These permissions may expose the user or the system to the least possible risk. They are automatically granted upon installation and cannot be revoked.

Dangerous: These permissions can expose private user data or allow control of the device. Therefore, explicit user approval is required to be granted and can be revoked by the user at any time. Dangerous permissions are illustrated in Figure 1.

Signature: While Android applications may exchange information through inter component communication, to facilitate apps of the same developer in sharing more information, Google introduced the signature permission, enabling automatic access to the same resources for application which are signed with the same certificate without user notification.

System: To allow applications which are shipped by the manufacturers, to have even more elevated privileges, *e.g.* to reboot the device, activate and deactivate system settings, Android provides the system permission.

Apparently, since voice assistants are usually developed by the manufacturers and/or OS vendors, they have signature and system level permissions. Hence, they can execute commands with higher privileges. Despite the fact that the new permission model signifficantly improves the security and privacy of Android, many attacks have already been reported that manage to bypass them exposing users to great risks [16], [17]. Nevertheless, Figure 1 indicates that in order for an app to access the microphone or to send a message etc., the app needs to be explicitly granted this permission. Finally, access to any other sensor beyond the ones reported in this figure are automatically granted and cannot be revoked.

III. Attacking voice assistants revisited

A. Threat model

In the proposed attacks, which portray the vulnerabilities of voice assistants, we assume that a user has been tricked into installing our malicious app in his/her device, a scenario which is very typical in such attacks [18], [19], [20]. In fact, an adversary is expected to use many illicit techniques that promote their apps in the market, or lure users by sending targeted messages to their contacts.

We consider the above a weak requirement, as in our work we follow a "zero-permission" approach. Similar to the work of Diao et al. [8], our app is seemingly harmless as it does not request access to any "dangerous" permission. Moreover, since a small number of the required permissions are "normal", they are automatically granted and cannot be revoked. The latter also means that any installed app can be updated to execute our attacks, without requesting any further user interaction. More specifically, application updates may take place at any time and especially in the cases where no extra "dangerous" permission is required, that imposes explicit action taken by the user, users have no actual control about a new "capability" an app may be shipped with. In fact, in most of the cases, the only permission we need in order to deliver our payload and launch an attack to a voice assistant is the Internet access permission, which is a normal permission. A few more permissions, *e.g.* vibration, that are required in some cases are also normal and they are only used to determine the timing of the attack and the proximity to other devices and to the user. The lack of permission dependencies in our attacks is crucial as one could use reflection techniques to fetch code from the Internet, *e.g.* in the form of a dex file and execute it, circumventing static code analysis [21], [22], [23] . Nonetheless, it should be outlined that many users may disregard the permissions that apps request [13], [14], [15], however, requesting many permissions, especially dangerous ones, facilitates the detection of malicious activity.

To this end, as a first step, we silently monitor the state of the target device in order to determine whether the user is in proximity. This can be easily achieved by effectively combining data such as time (part of the day), device acceleration (moving/still), device battery status (charging/discharging), or even utilizing light sensors (indoor/outdoor). Should the app infer that the device is left "unattended", it sends an update to the Command and Control (C&C) server declaring its availability.

B. Attack amplification

In the next paragraphs we present our attach flow, while at the same time we detail the drawbacks of current reported attacks and how they can be enhanced to become more lightweight, stealth and powerful. In this regard, we provide the pieces of a bigger puzzle which is going to be presented when we describe the possible attack scenarios.

Current state of the art attacks to voice assistants can be considered rather static, as the attacks are launched using mostly recorded audio files which are played by malicious apps to invoke voice assistants [8] or broadcasted by a static medium (*e.g.* a YouTube video) [9] for a more covert channel. While Diao et al. [8] imply the use of Text-to-Speech, it is not actually used due to the requirement for Internet connection. Moreover, Diao et al. suggested that the malicious app makes a call to the attackers phone to allow further interaction with the voice assistant, and more importantly to extract the necessary sensitive information from the victim, since this is the only way to access the microphone without requesting any sort of dangerous permission. However, to the best of our knowledge this attack vector does not exist anymore, as voice assistants cannot be triggered by audio by means of a voice call. Mobile OSes are now using voice cancellation techniques during voice calls, therefore audio from voice calls cannot be used as input to the microphone. While in this case it can be considered a defence mechanism, in reality it improves the quality of voice calls by drastically reducing external noise.

To overcome the need for making phone calls, which could also reveal the identity of the attacker, we propose to use the embedded Text-to-Speech technology. This choice enables our app to be rather lightweight, as there is no dependency for

	Normal	**Dangerous**	**Signature**	**System**
Assignee	System	User	System/App	System
Revocable	No	Yes	No	No
Risk level	Low	High	Critical	Critical
Access	Sensors/actions with low risk	Sensors/actions with high risk	Sensors/actions with high risk	Sensors/actions with critical risk
Example resource/ functionality	Accelerometer, Internet, Environemental sensors	Camera, Microphone, SMS, Storage, Calls, Location	Privileges of another app of the same developer	Reboot/encrypt device, change system settings

TABLE I
ANDROID PERMISSIONS LEVELS. THE FULL CATALOG OF DANGEROUS PERMISSIONS IS ILLUSTRATED IN FIGURE 1.

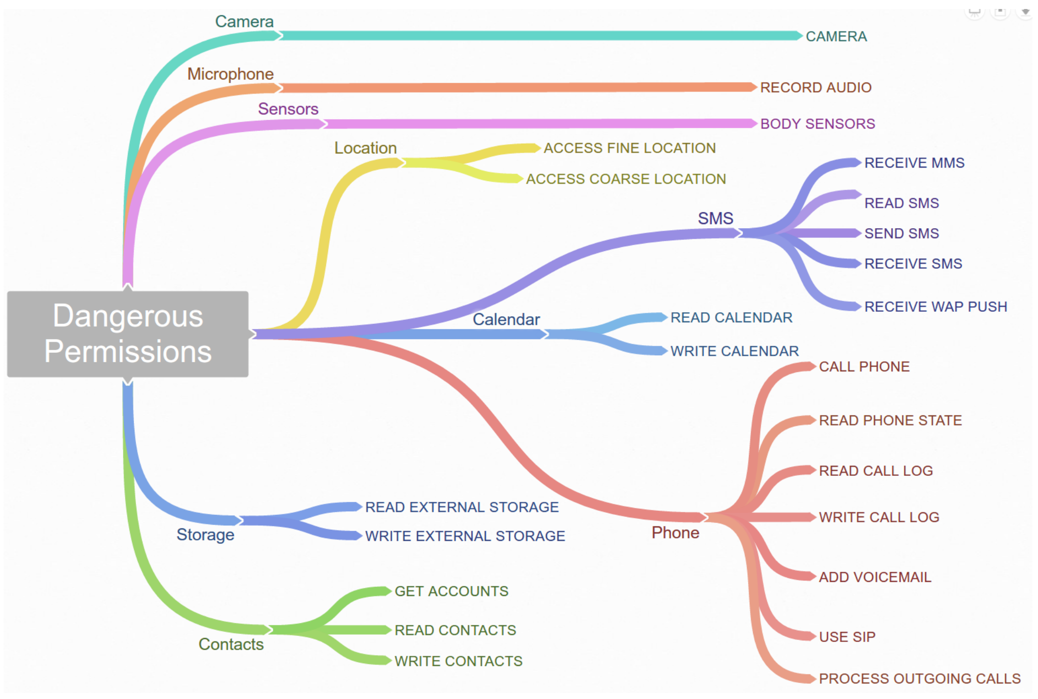

Fig. 1. Dangerous permission groups in Marshmallow.

prepared audio files, making our attack more “dynamic” since we can generate any voice command in demand. Contrary to the past, Text-to-Speech conversion is now performed locally without Internet dependencies. Nevertheless, in our model we use the Internet permission to exchange information between the attacker and the victim, since, as already discussed, it is automatically granted and cannot be revoked.

While we could have used direct communication of the victim with a typical C&C server as in the case of many malware with their bots, we opted for the use of Firebase, a Google powered cloud platform that incorporates real-time databases. The use of Firebase can be considered similar to the use of Facebook, Twitter, etc. by social botnets [24] to hide their C&C server and traffic. This practice is very common lately [25] as it manages to hide the malicious traffic from anti-malware mechanisms which consider the exchange of messages benign. In our case, we hide the communication using a legitimate channel of communication operated by Google, so that it would not raise an alert to an analyst.

Another twist in our attack is on the way we launch voice assistants. In Diao et al. [8] the researchers assume that due to the popularity of Google Now, the victim will most likely have it preinstalled in his/her device. This assumption may result in several problems as the authors also use implicit intents in their implementation. For instance, these attacks may not be launched if Google Now is not present. Or even due to the fact that more assistants are installed in a device, a screen would be presented to the user prompting him to select one of them, and thus rendering these attacks useless in both cases. To bypass this obstacle, in our implementation for each victim’s device we retrieve the applications’ package names of all running services. For this action no permission is required and we end up with a list of all the installed packages. Subsequently, from the list of the installed packages we are able to first determine which voice assistants are installed in each mobile device and secondly which permissions each voice assistant supports. This small, yet significant remark can be proposed as a way to launch attacks to “less popular” voice assistants software solutions, developed by other software vendors than the top tech giants, analyzed in this paper. After having defined both the targeted voice assistant and the “kind” of attack we want to issue, we use explicit intents, by specifying the targeted application’s package name, guaranteeing that the attack will be successful.

As previously discussed, attacks to voice assistants in the literature cannot extract all the information, as the output of the speaker is canceled in the microphone. To overcome this issue, we use native voice recognition methods of Android which, as reported by the authors[1], do not request any dangerous permission. More precisely utilizing Google Voice Recognition one can collect data from the audio channel, without users' consent and without requesting the use of microphone from the user, bypassing this way Android's permission system. This way, we automate sensitive data collection, by automatically recording voice assistants' results, consequently converting it to text format which is then stored to Firebase. This is accomplished by using an implicit intent to `action_recognise_speech` with `startActivityForResult`, which returns the voice in text without the need to request any dangerous permission.

To illustrate the aforementioned described attack scenarios, which can be considered as quite complex in terms of services discovery, attack orchestration and modalities of interaction, Figure 2 describes the steps of our attack flow.

Up to recently, voice assistants were able to perform helpful, yet limited in number, tasks. However, they are gradually increasing their capabilities as other applications have started enabling interaction with them. Therefore, voice assistants can perform far more tasks than just send emails and SMSs, make voice calls, as reported in the related work. For instance, they can post on social media (*e.g.* Facebook) or even perform orders, *e.g.* in the case of Alexa. These new capabilities are exploited in our proof of concept app, enabling it to perform a wide set of commands to victims, so as to achieve more fierce attacks.

C. Determining proximity

Due to the fact that malicious apps cannot interact or send information directly to the adversary, the actual threats that are discussed in current state of the art research involve attacks which are targeting devices independently. Certainly, the case of a YouTube video, reported in [10], can target millions of devices simultaneously, nonetheless, the attack targets each one and cannot chain the attack to infect others directly.

Our proof of concept exploits its proximity to other voice assistants or nearby devices to perform a wider range of attacks, creating an implicit machine-to-machine communication channel. To launch such attacks, our application first determines the timeframe and then the proximity to the user and to other devices. For the latter, we monitor various resources to determine whether the phone is still and out of the users' proximity. To this end, apart from the obvious use of accelerometer, we monitor the presence of a power jack, luminosity and grip sensor. Should the device be charging, the user is assumed to be far from the device and not paying attention to it. Similarly, the measurements from a grip sensor can indicate whether a user is holding the device, whereas the luminosity sensor can help us determine whether a device is located in a dark place *e.g.* left in a room unattended, or inside a bag. Finally, we also monitor the proximity of wearables. This is an additional input which can be used to determine whether a user is in proximity or *e.g.* sleeping, so that he is least expected to listen to the attack. It is worth noticing that modern smartwatches can independently execute voice commands, since voice assistants are currently enabled in Android wearables. In this research however, we regard the linked wearable and the infected device's wearables as the same entity.

To monitor the proximity to other devices, we use four methods, illustrated in Figure 3. To overcome the restriction of requesting location permission, which is considered dangerous (requires "dangerous" permissions), we first utilize Wi-Fi and Wi-Fi P2P. While getting hardware identifiers from Wi-Fi and Bluetooth scans demand location permission as of Marshmallow, collecting the name and the MAC address of the connected Wi-Fi node can be achieved with normal permissions and can serve as a good sieve. Therefore, our app collects the name of the connected Wi-Fi and the MAC address of the corresponding node to determine which already infected devices might be in proximity. Additionally, we can use Wi-Fi P2P, formerly known as Wi-Fi Direct, which enables devices to create ad hoc wireless networks. Notably, scanning for hardware identifiers using Wi-Fi P2P does not require location permission, as reported by the authors,[2] allowing the tracking of users' whereabouts within a radius of 100m [26]. For more accurate results we use vibration (requesting a normal "Vibrate" permission) and light sensing. In the former method, we trigger a vibration pattern to one of them and test whether it was detected by the accelerometers of the other, to determine whether the two devices are placed closely. In the latter method, if the devices are placed appropriately; one facing up, the other one facing down, we initiate a sequence of light beams and test whether they can be sensed from the other's luminosity sensor. The presence of a nearby wearable device can also be detected, this time directly through the Android SDK, requiring no permissions.

D. Attack scenarios

Now that all the pieces of the puzzle have been laid out, in this part we start putting them in place. Our targets are the most well-known and widely used voice assistants, namely Google Assistant, Siri, Cortana, Alexa and S Voice, each of which has different capabilities, as seen in Table II. In principle, these systems are invoked by speech, whenever a user says a catch phrase like "OK Google", "Hey Siri" etc. However, Google Assistant, Cortana and S Voice can also be launched through intents. Since intents are a native mechanism of Android, no dangerous permission needs to be granted to the malicious application. Even more interestingly, intent parameters are string variables, which do not have to be hard-coded and can be easily fetched from an Internet service, in our case Firebase, making the attack untraceable in static analysis. This means that one can launch the voice assistant remotely and issue arbitrary commands through an app. More importantly, this allows our malicious application to send commands which need more than one step for their execution. For instance, some voice assistants may type the

[1]Acknowledged by Google in 216234 report.

[2]Acknowledged by Google in 216235 report.

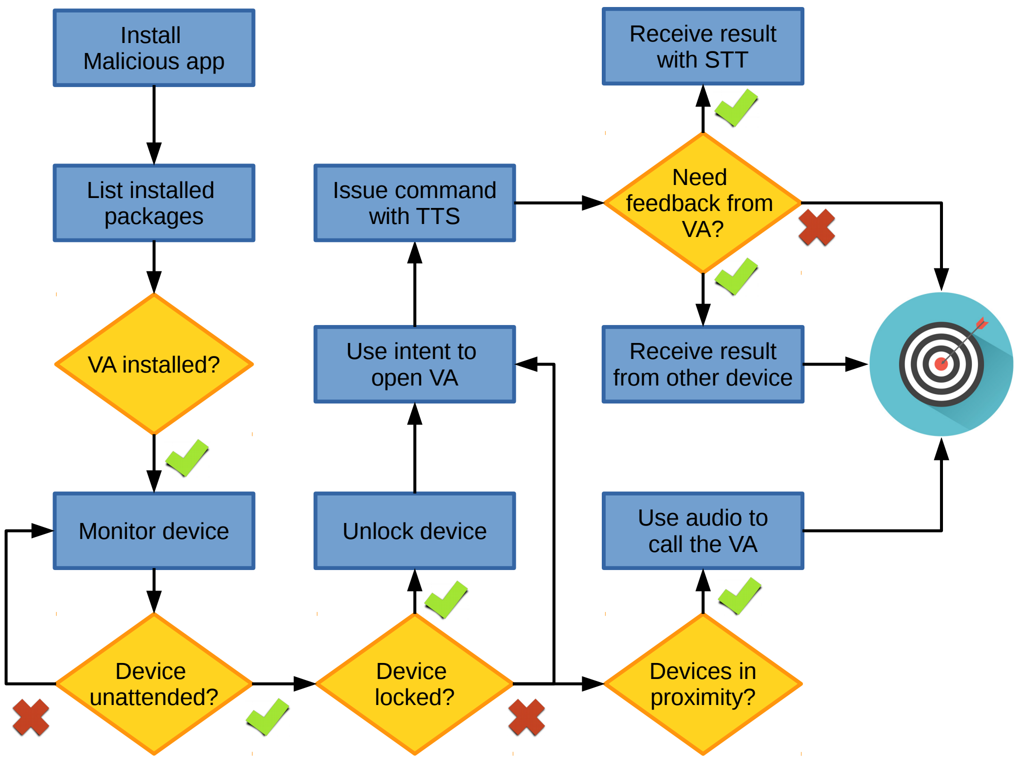

Fig. 2. Overview of the attack flow.

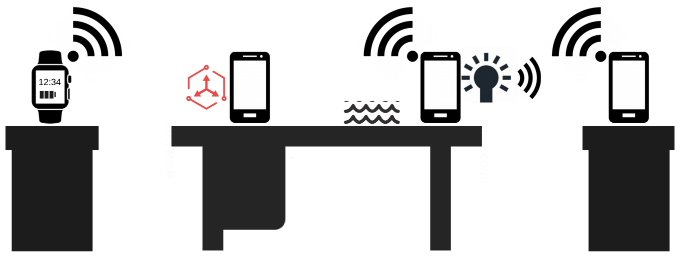

Fig. 3. Monitoring device proximity. If two devices lie on the same table one can sense the vibration of the other, or if properly positioned, sense the light beams. Other proximity methods such as usage of Wi-Fi and Wi-Fi P2P can also be used.

requested message but wait for an additional vocal confirmation from the user to send it. While one could record such commands and send them to victims either packed in the application upon installation, or through a covert channel, the most lightweight solution is to use Android's Text-To-Speech (TTS) functionality. This functionality requires no permission to be executed, so the user will not be notified about this activity.

In the least sinister case, an adversary can request the infected device to call a phone number to premium services, leading to additional charges, or simply to call his/her phone to eavesdrop the victim. Note that the call could also include video, *e.g.* using Hangouts teleconference issuing a command like "Video call with Alice". Additionally, the attacker can request the victim to send an SMS to a premium number, or a text message containing his/her location.

Similarly, the adversary can use the Voice Assistants' integration with third party apps, *e.g.* social network apps, to post disgraceful or harmful items on the victim's profile. For example, in our tests we managed to send information using the WhatsApp app by issuing a "Send a WhatsApp message to Alice" command. Further extending the attacks, one could also command the Voice Assistant to browse an infected with malware web-page in order to gain persistence, as already reported in the literature.

To extract further private information, the attacker needs to gain microphone access to its victim. Contrary to the approach of current attacks, we use Speech-to-Text (STT). As already mentioned in the previous section, the adversary does not need to request any dangerous permission, and can record information of approximately 10 seconds from the victim, which is automatically transformed into text and can be easily sent to a cloud backend. Obviously, if the answer is expected to last more seconds, the adversary can repeat the process with the necessary delay. Having knowledge about devices being in proximity can further help us bypass the "Voice Cancellation" mechanisms that reside in a specific device, since one device can be used as a transmitter and another device can be used as the receiver. Additionally, this makes the attack more stealth and more difficult to be prevented.

The adversary may now extract information such as appointments from the calendar, or recent messages. The latter is really important since it can be used not only to read personal messages, but to bypass two factor authentication. To clarify the latter, the reader can consider the following use case scenario. Most messenger applications that use two factor authentication typically use the device MAC address; which can be easily found with a sniffer, and a validation SMS which is sent to the device. Therefore, Malory can easily set the MAC address to the desired one in a device she controls and request the SMS authentication message from the service. This message is received from the victim's device and can be accessed by the voice assistant using a command to read the last SMS, granting full access to the victim's account.

Due to the penetration of services like Google Now, the adversary could also read the victim's bills, or flight itinerary. Moreover, using the elevated permissions of S Voice (system permissions), one could even change the status of Location and Mobile Hotspot by switching them on or off, a functionality that an application without signature permissions cannot achieve, as these actions require "changing system settings", signature level permissions.

Moving a step further, compromised devices can be used to further exploit victims, by attacking other devices in proximity, regardless of whether they are infected by our malicious app, as it is illustrated in Figure 4. Attackers now have the advantage that since the arbitrary voice command originates from another device, voice cancellation mechanisms cannot be applied. Targets also vary, as there is a plethora of devices having voice assistants, leading to greater exposure. An obvious attack is against other smartphones, smartwatches, personal computers, or even smartTVs. Note that in the case of a smartwatch, since most likely it is linked to a smartphone, two smartphones are not needed to be in proximity, as smartwatches may act as a median, further extending the impact radius.

In this context, a different attack scenario is to force compromised devices in performing purchases through nearby Alexa powered devices. In our tests we ordered Alexa through Echo using voice commands to buy for us several items from Amazon. It is important to note that Alexa by default does not provide any voice recognition or parental control, so all orders are automatically processed. Moreover, Alexa is a voice assistant which is actually designed from scratch to perform purchases and is being gradually equipped with third-party applications, which allow her to perform orders, or call taxis from affiliated companies.

Moving to another target, we used the compromised device to attack a Cortana powered device. Apart from the obvious smartphone target, we used it to attack a desktop computer, since Cortana is preinstalled in Windows 10. Launching Cortana through voice, we were able to navigate the system to arbitrary web pages, control settings (*e.g.* enable/disable Wi-Fi), create reminders, events, alarms, find location, send emails, make calls and send SMSs through linked smartphones regardless of their proximity.

As a smartphone app, Cortana by default is trained to respond only to her "master's" voice, something that is also relevant for Siri and S Voice, but not the default case for the latter. In these cases, the "trusted voices", as they are referred, can be bypassed using two methods. In the more obvious scenario, we used recordings of the user's voice and replayed them using the compromised device's speakerphones. While this approach is straightforward, it is cumbersome to get all the needed wording from the victim to cut and choose the necessary commands to launch the attack. A feasible alternative for the attacker is to use a "trained" voice engine. In this case, the attacker would take a good sample of the victim's voice and train the voice engine to mimic his/her voice. Note that since the set of commands that we want to issue is quite limited, the actual sample that is needed is also reduced. It is worth mentioning that Siri, Google Assistant and S Voice can be triggered even when a smartphone is securely locked (e.g. pin/pattern lock), if the voice is trusted. Interestingly, Cortana for Desktop computers has an option to be launched from "anyone" who says "Hey Cortana", which is also the case of Alexa Echo.

Finally, as shown in Table II, voice assistants can be connected to even more devices, varying from vehicles to smart appliances, to allow them partial control. In this regard, taking control of a voice assistant, may result in also controlling the linked devices. Knowing in advance whether someone has smart appliances in his/her home sounds as a far fetched scenario, the truth however is that it is quite simple to acquire such knowledge. In the case of Android, one can extract a list of all the installed applications without stumbling upon any dangerous permission. Due to their "smartness", the connected devices are expected to have the respective apps installed in the user's device, informing the adversary of their presence.

IV. APPLICABILITY

All reported attacks have been tested and responsibly disclosed to Google, while in the cases of other voice assistants, their corresponding companies have also been informed. Notably, all these attacks can be applied in Android AOSP which at the time of writing is Nougat.

In all our tests, the attacks where successful both in the cases of unattended devices and also in cases of locked devices (simple lock, not secure lock). While not all users lock their devices, even if they do so, applications may use the normal permissions `WAKE_LOCK` and `DISABLE_KEYGUARD`, as well as the activity flag `FLAG_KEEP_SCREEN_ON` to prevent or prolong the screen lock to timely launch a Voice Assistant. Finally, even in the cases of securely locked devices (pin or pattern lock) it should be noted that all Android smartwatches can be considered as trusted devices by the users of the connected smartphones, therefore, if they are in proximity the devices do not actually securely lock. Therefore, even in the "most difficult" for the attacker case, when a device is securely locked, if the user is not close but in proximity to the mobile device e.g. 10m or another room, the device will not be securely locked and the attack can be executed without the user being able to notice it.

To further enhance the efficacy of our attacks we also managed to get our malicious app accepted in Google Play. This is rather significant since published apps are being tested by Bouncer[3], a system which analyses apps for malicious functionality, mostly using dynamic analysis [27], indicating that this attack vector is not being considered. However, as already discussed, the aforementioned attack scenarios utilize

[3]http://googlemobile.blogspot.gr/2012/02/android-and-security.html

Voice Assistant	Siri	Google Assistant	Alexa	Cortana	S Voice
Voice Authentication	✓	✓	Prob No		✓
Activation methods	V	V,I	V	V,I	V,I
Online dependency	✓				
Access device settings	✓	✓	Alexa Skills		✓
Post Private Information	✓	✓		✓	✓
Purchases	✓	To come	✓		✓
Calls/SMS*	✓	✓	Skills	✓	✓
3rd party apps	✓	✓	✓	✓	Only launch
Permission Levels	System	System	System	System/ Dangerous	System
Connectivity	Car (via CarPlay), Apple Watch, PC	Wearables, TV, Auto, Home	Home, Smart Devices	PC, Xbox One	-
Always on	✓		✓		
Wake up in secure lock	✓	✓			✓

TABLE II
CAPABILITIES OF VOICE ASSISTANTS. ACTIVATION METHODS, APART FROM (V)OICE, MAY ALSO INCLUDE (I)NTENTS WHICH CAN BE EXPLOITED PROGRAMMATICALLY.

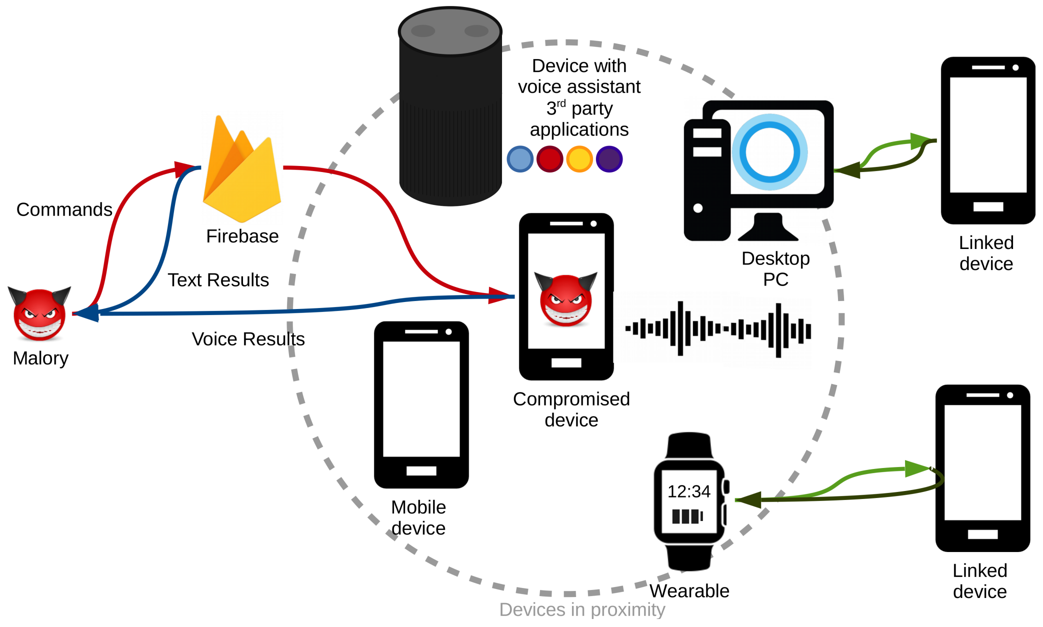

Fig. 4. Compromising devices in proximity.

mechanisms that are unlikely to be traced. As a result, the attacks described in this paper apply to smartphones of all Android versions in the market, having as precondition that at least one voice assistant is present.

To summarize the applicability of the attacks to voice assistants we argue that our attacks are able to affect voice assistants that are incorporated in a large variety of smart devices, whose number is continuously growing in the IoT era. These devices include the highly widespread smartphones and tablets, the personal computers, modern wearables, smart-home devices and TVs and we might shortly find them in cars and vehicles incorporating smartphone OSes. Their exploitation might initiate from applications that, as shown in this study, are quite straightforward to be used for malicious reasons, however it does not stop there. Voice assistants are software entities with enormous capabilities and permissions, whose potential might be exploited by web-pages or even other malicious people, having physical access to a device. This kind of research is beyond the scope of this paper, however it is easily provable that either websites or people can issue voice commands in assistants, especially in the cases where the assistants are not able to determine the identity of their "owner".

V. CONCLUSIONS

Voice assistants are gradually getting to a momentum, since they are being preinstalled in many devices, and they are getting smarter gaining support from third parties to interact with an increasing number of applications. Therefore, they are radically changing user behaviour and expectations as they provide new levels of convenience and an enhanced user experience. However, they suffer from an inherent flaw: up till now, they depended solely on the vocal modality of interaction. As demonstrated in this work, one can easily control and manipulate the voice commands remotely, issuing arbitrary commands which can greatly expose the users. Nonetheless, we are not able to fully understand these risks to their full extent. One reason is that voice assistants are backed by artificial intelligence making them more extensible. However, common practice has shown that such systems can be easily manipulated [28]. Finally, it is apparent that countering such attacks is not straightforward. Voice cancellation and recognition might seem probable solutions, yet they cannot be considered a panacea, due to the way voice assistants can be launched and the fact that one can chain the attacks from one device to another.

We argue that in order to provide defense mechanisms against many of these attacks, the underlying operating system, in this case Android, firstly needs to decouple voice input and output. Having simultaneously a number apps using voice input and output has already been shown to expose users' security and privacy, therefore, there is a definite need from the OS side to consider both microphone and speaker as a unique communication channel and identify apps that try to use both flows (input and output) when only one is granted by the user. This approach has been originally proposed by AuDroid [29] and some of the concepts have already been deployed in AOSP. Nevertheless, it cannot identify all the information flows discussed in this work, as they can also involve other devices. Moreover, the issue is complicated by the use of intents and the fact that Voice Assistants are constantly activated, which invalidates the assumption of only one application using the communication channel. A rather simplistic, yet powerful defense mechanism against the attacks that target voice assistant would be to utilize biometrics in order to identify the actual device owners and enable or disable the function of these assistant accordingly.

Finally, based on the permissions the user is presented, regardless of whether they are requested on installation or runtime, one expects that the installed apps will not use the microphone if s/he does not explicitly grant the permission. Allowing applications to arbitrarily access the microphone *e.g.* through intents, is far more than risky and as detailed in this work can have more implications than simply spying on users' conversations.

ACKNOWLEDGMENTS

This work was supported by the European Commission under the Horizon 2020 Programme (H2020), as part of the *OPERANDO* project (Grant Agreement no. 653704) and is based upon work from COST Action *CRYPTACUS*, supported by COST (European Cooperation in Science and Technology).

REFERENCES

[1] J. Jiang, A. Hassan Awadallah, R. Jones, U. Ozertem, I. Zitouni, R. Gurunath Kulkarni, and O. Z. Khan, "Automatic online evaluation of intelligent assistants," in *Proceedings of the 24th International Conference on World Wide Web*, ser. WWW '15. International World Wide Web Conferences Steering Committee, 2015, pp. 506–516.

[2] J. Kiseleva, K. Williams, J. Jiang, A. Hassan Awadallah, A. C. Crook, I. Zitouni, and T. Anastasakos, "Understanding user satisfaction with intelligent assistants," in *Proceedings of the 2016 ACM on Conference on Human Information Interaction and Retrieval*, ser. CHIIR '16. ACM, 2016, pp. 121–130.

[3] Mindmeld, "Intelligent voice assistants: User adoption survey results Q1 2016," http://info.mindmeld.com/survey2016q1.html.

[4] T. Mozer, *Speech's Evolving Role in Consumer Electronics... From Toys to Mobile*. Springer New York, 2013, pp. 23–34.

[5] S. Pichai, "Google I/O 2016 keynote," https://www.youtube.com/watch?v=862r3XS2YB0.

[6] I. Guy, "Searching by talking: Analysis of voice queries on mobile web search," in *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 2016, pp. 35–44.

[7] Y. Jang, C. Song, S. P. Chung, T. Wang, and W. Lee, "A11y attacks: Exploiting accessibility in operating systems," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 103–115.

[8] 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 *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*. ACM, 2014, pp. 63–74.

[9] T. Vaidya, Y. Zhang, M. Sherr, and C. Shields, "Cocaine noodles: exploiting the gap between human and machine speech recognition," in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, 2015.

[10] N. Carlini, P. Mishra, T. Vaidya, Y. Zhang, M. Sherr, C. Shields, D. Wagner, and W. Zhou, "Hidden voice commands," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016.

[11] C. Kasmi and J. L. Esteves, "Iemi threats for information security: Remote command injection on modern smartphones," *IEEE Transactions on Electromagnetic Compatibility*, vol. 57, no. 6, pp. 1752–1755, 2015.

[12] SnoopWall, "Flashlight apps threat assessment report," http://www.snoopwall.com/wp-content/uploads/2015/02/Flashlight-Spyware-Report-2014.pdf, 2014.

[13] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proceedings of the Eighth Symposium on Usable Privacy and Security*. ACM, 2012, p. 3.

[14] P. G. Kelley, S. Consolvo, L. F. Cranor, J. Jung, N. Sadeh, and D. Wetherall, "A conundrum of permissions: installing applications on an android smartphone," in *Financial Cryptography and Data Security*. Springer, 2012, pp. 68–79.

[15] R. Balebako, J. Jung, W. Lu, L. F. Cranor, and C. Nguyen, "Little brothers watching you: Raising awareness of data leaks on smartphones," in *Proceedings of the Ninth Symposium on Usable Privacy and Security*. ACM, 2013, p. 12.

[16] Y. Fratantonio, C. Qian, S. Chung, and W. Lee, "Cloak and Dagger: From Two Permissions to Complete Control of the UI Feedback Loop," in *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, San Jose, CA, May 2017.

[17] E. Alepis and C. Patsakis, "Trapped by the ui: The android case," in *Proceedings of the 20th International Symposium on Research in Attacks, Intrusions and Defenses*. Springer, 2017, (To appear).

[18] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2011, pp. 3–14.

[19] T. Vidas, D. Votipka, and N. Christin, "All your droid are belong to us: a survey of current android attacks," in *Proceedings of the 5th USENIX conference on Offensive technologies*. USENIX Association, 2011, pp. 10–10.

[20] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan, "Android security: a survey of issues, malware penetration, and defenses," *IEEE communications surveys & tutorials*, vol. 17, no. 2, pp. 998–1022.

[21] S. Poeplau, Y. Fratantonio, A. Bianchi, C. Kruegel, and G. Vigna, "Execute this! analyzing unsafe and malicious dynamic code loading in android applications." in *NDSS*, vol. 14, 2014, pp. 23–26.

[22] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan, "Android security: A survey of issues, malware penetration, and defenses," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 2, pp. 998–1022, 2015.

[23] M. Backes, S. Bugiel, E. Derr, P. McDaniel, D. Octeau, and S. Weisgerber, "On demystifying the android application framework: Re-visiting android permission specification analysis," in *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, pp. 1101–1118.

[24] E. J. Kartaltepe, J. A. Morales, S. Xu, and R. Sandhu, "Social network-based botnet command-and-control: emerging threats and countermeasures," in *International Conference on Applied Cryptography and Network Security*. Springer, 2010, pp. 511–528.

[25] J.-I. Boutin, "Turla's watering hole campaign: An updated firefox extension abusing instagram," https://www.welivesecurity.com/2017/06/06/turlas-watering-hole-campaign-updated-firefox-extension-abusing-instagram/.

[26] E. Alepis and C. Patsakis, "There's Wally! Location tracking in Android without permissions," in *International Conference on Information Systems Security and Privacy*, 2017.

[27] J. Oberheide and C. Miller, "Dissecting the android bouncer," in *SummerCon*, 2012.

[28] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 427–436.

[29] G. Petracca, Y. Sun, T. Jaeger, and A. Atamli, "Audroid: Preventing attacks on audio channels in mobile devices," in *Proceedings of the 31st Annual Computer Security Applications Conference*, ser. ACSAC 2015. ACM, 2015, pp. 181–190.

APPENDIX A
VOICE ASSISTANT PERMISSIONS

To understand the capacities of the voice assistants and the actions they are allowed to perform, the Table III provides an overview of the permissions they are granted in Android. Note that while some documentations describe the command sets that they are allowed to execute, their permissions allow them to perform far too many tasks if exploited.

Google Now (Assistant)	Samsung S-Voice	Microsoft Cortana
com.google.android.googlequicksearchbox.permission.FINISH_GEL_ACTIVITY	android.permission.WRITE_EXTERNAL_STORAGE	android.permission.INTERACT_ACROSS_USERS_FULL
com.google.android.apps.now.CURRENT_ACCOUNT_ACCESS	android.permission.MODIFY_AUDIO_SETTINGS	android.permission.DOWNLOAD_WITHOUT_NOTIFICATION
com.google.android.googlequicksearchbox.permission.C2D_MESSAGE	android.permission.ACCESS_NOTIFICATION_POLICY	android.permission.USE_CREDENTIALS
com.google.android.c2dm.permission.RECEIVE	com.sec.android.app.twdvfs.DVFS_BOOSTER_PERMISSION	android.permission.BLUETOOTH
android.permission.GLOBAL_SEARCH	com.samsung.voiceserviceplatform.permission.BACKUP_RESTORE	android.permission.BLUETOOTH_ADMIN
android.permission.READ_CONTACTS	com.sec.android.settings.permission.SOFT_RESET	android.permission.INTERNET
android.permission.WRITE_CONTACTS	com.samsung.mdl.radio.permission.READ_RADIO_STORAGE	android.permission.CHANGE_WIFI_STATE
com.android.browser.permission.READ_HISTORY_BOOKMARKS	com.sec.imsservice.PERMISSION	android.permission.WRITE_SETTINGS
com.android.browser.permission.WRITE_HISTORY_BOOKMARKS	com.sec.imsservice.READ_IMS_PERMISSION	android.permission.SYSTEM_ALERT_WINDOW
android.permission.ACCESS_NETWORK_STATE	com.sec.imsservice.WRITE_IMS_PERMISSION	com.android.alarm.permission.SET_ALARM
android.permission.INTERNET	com.samsung.svoice.sync.READ_DATABASE	cyanogenmod.permission.MANAGE_ALARMS
com.google.android.launcher.permission.CONTENT_REDIRECT	com.samsung.svoice.sync.WRITE_DATABASE	cyanogenmod.permission.READ_ALARMS
com.google.android.launcher.permission.READ_SETTINGS	com.samsung.svoice.sync.ACCESS_SERVICE	cyanogenmod.permission.MODIFY_NETWORK_SETTINGS
com.google.android.launcher.permission.WRITE_SETTINGS	android.Manifest.permission.MEDIA_CONTENT_CONTROL	cyanogenmod.permission.MODIFY_SOUND_SETTINGS
com.google.android.launcher.permission.RECEIVE_LAUNCH_BROADCASTS	android.permission.MEDIA_CONTENT_CONTROL	android.permission.CAPTURE_AUDIO_HOTWORD
com.google.android.launcher.permission.RECEIVE_FIRST_LOAD_BROADCAST	android.permission.CALL_PRIVILEGED	android.permission.REBOOT
android.permission.USE_CREDENTIALS	com.google.android.providers.gsf.permission.READ_GSERVICES	com.android.launcher.permission.INSTALL_SHORTCUT
android.permission.MANAGE_ACCOUNTS	com.samsung.android.scloud.backup.lib.read	com.android.launcher.permission.UNINSTALL_SHORTCUT
android.permission.WRITE_SETTINGS	com.samsung.android.scloud.backup.lib.write	com.android.launcher.permission.READ_SETTINGS
com.google.android.providers.settings.permission.READ_GSETTINGS	android.permission.GET_TASKS	com.android.launcher3.permission.READ_SETTINGS
com.google.android.providers.settings.permission.WRITE_GSETTINGS	com.samsung.permission.SBEAM_SETTINGS	com.google.android.launcher.permission.READ_SETTINGS
com.google.android.providers.gsf.permission.READ_GSERVICES	android.permission.CHANGE_NETWORK_STATE	com.huawei.android.launcher.permission.READ_SETTINGS
com.google.android.voicesearch.SHORTCUTS_ACCESS	android.permission.WRITE_SYNC_SETTINGS	android.permission.ACCESS_FINE_LOCATION
com.google.android.voicesearch.ACCESS_SETTINGS	android.permission.ACCESS_NETWORK_STATE	android.permission.ACCESS_COARSE_LOCATION
com.android.browser.permission.PRELOAD	android.permission.ACCESS_FINE_LOCATION	android.permission.DISABLE_KEYGUARD
com.google.android.ears.permission.WRITE	android.permission.ACCESS_COARSE_LOCATION	android.permission.WAKE_LOCK
android.permission.BROADCAST_STICKY	android.permission.READ_SYNC_SETTINGS	android.permission.VIBRATE
android.permission.MEDIA_CONTENT_CONTROL	android.permission.INTERNET	android.permission.READ_PHONE_STATE
android.permission.INTERACT_ACROSS_USERS	android.permission.RECORD_AUDIO	cyanogenmod.permission.THIRD_PARTY_KEYGUARD
com.google.android.googlequicksearchbox.permission.PAUSE_HOTWORD	android.permission.VIBRATE	cyanogenmod.permission.LIVE_LOCK_SCREEN_MANAGER_ACCESS_PRIVATE
android.permission.GET_PACKAGE_SIZE	android.permission.READ_PHONE_STATE	android.permission.READ_SYNC_SETTINGS
android.permission.ACCESS_WIFI_STATE	android.permission.BLUETOOTH	android.permission.WRITE_SYNC_SETTINGS
android.permission.ACCESS_NOTIFICATION_POLICY	android.permission.BLUETOOTH_ADMIN	android.permission.AUTHENTICATE_ACCOUNTS
android.permission.CHANGE_WIFI_STATE	android.permission.BROADCAST_STICKY	android.permission.MANAGE_ACCOUNTS
android.permission.BLUETOOTH_ADMIN	com.android.alarm.permission.SET_ALARM	com.google.android.c2dm.permission.RECEIVE
android.permission.CHANGE_NETWORK_STATE	com.sec.android.app.clockpackage.permission.READ_ALARM	android.permission.ACCESS_WIFI_STATE
android.permission.READ_EXTERNAL_STORAGE	com.sec.android.app.clockpackage.permission.WRITE_ALARM	android.permission.ACCESS_NETWORK_STATE
android.permission.RECEIVE_BOOT_COMPLETED	android.permission.READ_LOGS	android.permission.GET_ACCOUNTS
android.permission.WAKE_LOCK	com.sec.android.widgetapp.q1_penmemo.permission.READ	android.permission.READ_CONTACTS
com.google.android.gms.permission.ACTIVITY_RECOGNITION	com.sec.android.widgetapp.q1_penmemo.permission.WRITE	android.permission.READ_PROFILE
android.permission.READ_PROFILE	android.permission.CALL_PHONE	android.permission.RECEIVE_SMS
com.google.googlenav.friend.permission.OPT_IN	android.permission.READ_CONTACTS	com.google.android.c2dm.SEND
com.google.android.apps.maps.permission.PREFETCH	android.permission.WRITE_CONTACTS	android.permission.RECEIVE_BOOT_COMPLETED
android.permission.ACCESS_COARSE_LOCATION	android.permission.ACCESS_WIFI_STATE	com.google.android.gms.permission.ACTIVITY_RECOGNITION
android.permission.ACCESS_FINE_LOCATION	android.permission.CHANGE_WIFI_STATE	android.permission.CALL_PHONE
android.permission.BLUETOOTH	android.permission.WAKE_LOCK	android.permission.WRITE_EXTERNAL_STORAGE
android.permission.CALL_PHONE	android.permission.READ_SMS	android.permission.WRITE_CALENDAR
android.permission.CALL_PRIVILEGED	android.permission.WRITE_SMS	android.permission.READ_CALENDAR
android.permission.GET_ACCOUNTS	android.permission.SEND_SMS	android.permission.SEND_SMS
android.permission.MODIFY_AUDIO_SETTINGS	android.permission.RECEIVE_SMS	android.permission.READ_SMS
android.permission.READ_CALENDAR	android.permission.RECEIVE_MMS	android.permission.WRITE_SMS
android.permission.READ_CALL_LOG	android.permission.RECEIVE_WAP_PUSH	android.permission.PROCESS_OUTGOING_CALLS
android.permission.READ_SMS	com.android.mms.permission.RECEIVE_MESSAGES_INFORMATION	android.permission.READ_EXTERNAL_STORAGE
android.permission.READ_PHONE_STATE	com.sec.mms.permission.RECEIVE_MESSAGES_INFORMATION	android.permission.CHANGE_NETWORK_STATE
android.permission.READ_SYNC_SETTINGS	android.permission.READ_CALENDAR	com.microsoft.bing.dss.async
android.permission.RECORD_AUDIO	android.permission.WRITE_CALENDAR	android.permission.RECORD_AUDIO
android.permission.SEND_SMS	com.sec.android.app.calendar.permission.READ_CALENDAR_SETTINGS	android.permission.GET_TASKS
android.permission.VIBRATE	android.permission.WRITE_SETTINGS	
android.permission.GET_TASKS	android.permission.RECEIVE_BOOT_COMPLETED	
android.permission.REAL_GET_TASKS	android.permission.READ_CALL_LOG	
android.permission.WRITE_CALENDAR	com.infraware.provider.SNoteProvider.permission.READ	
android.permission.WRITE_EXTERNAL_STORAGE	com.infraware.provider.SNoteProvider.permission.WRITE	
android.permission.WRITE_SMS	samsung.android.snoteprovider.permission.READ	
com.android.alarm.permission.SET_ALARM	samsung.android.snoteprovider.permission.WRITE	
com.google.android.apps.googlevoice.permission.AUTO_SEND	samsung.snote.permission.MEMO_CONTROL	
com.google.android.gm.permission.AUTO_SEND	com.samsung.android.app.notes.READ	
com.google.android.gm.permission.READ_GMAIL	com.samsung.android.app.notes.permission.C2D_MESSAGE	
com.google.android.googleapps.permission.GOOGLE_AUTH.cp	com.samsung.android.memo.READ	
com.google.android.googleapps.permission.GOOGLE_AUTH	com.samsung.android.memo.WRITE	
com.google.android.hangouts.START_HANGOUT	com.samsung.android.intent.action.MEMO_SERVICE	
com.google.android.voicesearch.AUDIO_FILE_ACCESS	com.android.browser.permission.READ_HISTORY_BOOKMARKS	
android.permission.CAMERA	com.sec.android.settings.myplace.permission.READ_DATA	
android.permission.FLASHLIGHT	com.sec.android.settings.myplace.permission.WRITE_DATA	
android.permission.DOWNLOAD_WITHOUT_NOTIFICATION	com.samsung.helphub.permission.HELP	
android.permission.MANAGE_VOICE_KEYPHRASES	com.wssnps.permission.COM_WSSNPS	
com.android.chrome.PRERENDER_URL	com.samsung.musicplus.provider.permission.READ_MUSICPROVIDER	
android.permission.CAPTURE_AUDIO_HOTWORD	com.samsung.musicplus.provider.permission.WRITE_MUSICPROVIDER	
android.permission.STOP_APP_SWITCHES	com.samsung.android.internal.intelligence.useranalysis.permission.READ_PLACE	
android.permission.STATUS_BAR	com.samsung.android.internal.intelligence.useranalysis.permission.WRITE_PLACE	
android.permission.SET_WALLPAPER	com.sec.android.provider.logsprovider.permission.READ_LOGS	
android.permission.SET_WALLPAPER_HINTS	android.permission.DISABLE_KEYGUARD	
android.permission.BIND_APPWIDGET	com.sec.voice.permission.RECEIVE	
com.android.launcher.permission.INSTALL_SHORTCUT	android.permission.WRITE_SECURE_SETTINGS	
android.permission.USE_FINGERPRINT	android.permission.INSTALL_PACKAGES	
	com.samsung.android.providers.context.permission.WRITE_USE_APP_FEATURE_SURVEY	
	android.permission.READ_EXTERNAL_STORAGE	
	com.samsung.alwaysmicon.permission.READ_CONTENT	

TABLE III
THE ANDROID PERMISSIONS OF VOICE ASSISTANTS.