Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones

Roman Schlegel
City University of Hong Kong
sschlegel2@student.cityu.edu.hk

Kehuan Zhang, Xiaoyong Zhou, Mehool Intwala, Apu Kapadia, XiaoFeng Wang
Indiana University Bloomington
{kehzhang, zhou, mintwala, kapadia, xw7}@indiana.edu

Abstract

We explore the threat of smartphone malware with access to on-board sensors, which opens new avenues for illicit collection of private information. While existing work shows that such "sensory malware" can convey raw sensor data (e.g., video and audio) to a remote server, these approaches lack stealthiness, incur significant communication and computation overhead during data transmission and processing, and can easily be defeated by existing protections like denying installation of applications with access to both sensitive sensors and the network. We present Soundcomber, a Trojan with few and innocuous permissions, that can extract a small amount of targeted private information from the audio sensor of the phone. Using targeted profiles for context-aware analysis, Soundcomber intelligently "pulls out" sensitive data such as credit card and PIN numbers from both tone- and speech-based interaction with phone menu systems. Soundcomber performs efficient, stealthy local extraction, thereby greatly reducing the communication cost for delivering stolen data. Soundcomber automatically infers the destination phone number by analyzing audio, circumvents known security defenses, and conveys information remotely without direct network access. We also design and implement a defensive architecture that foils Soundcomber, identify new covert channels specific to smartphones, and provide a video demonstration of Soundcomber.

1 Introduction

Today's mobile handsets are becoming full-fledged computing platforms capable of supporting complete operating systems, complicated applications and software development toolkits. With this technological revolution, however, come new security and privacy challenges. Like their PC counterparts, smartphones are no exception to the plague of data-stealing malware and recently there have been a number of incidents[1,2] and proofs-of-concept,[3,4] illustrating that smartphone malware is indeed a credible threat. The presence of unique sensors on these mobile platforms opens even more avenues for illicit collection of private user data. For example, a Trojan with access to the video camera [15] or microphone can tape a user's phone conversations and send the recording to other parties, which enables remote surveillance. Industry and academia have taken serious note of such threats, which we refer to as *sensory malware*. Newly released smartphone OSes all offer security protections: as an example, Google's Android separates different applications with Java virtual machines to mediate the interactions among them according to security policies. Anti-virus companies are moving their products to the mobile platform, e.g., McAfee's VirusScan Mobile[5] and Symantec's Norton Smartphone Security.[6] New security services [5] have been proposed to control installing untrusted software with dangerous security configurations, for example, applications that request both access to the microphone and an Internet connection, and to control communication between applications [10].

Such protections seem to be reasonably effective against phone-borne malware, whose complexity and stealthiness

[1] http://www.sophos.com/blogs/gc/g/2010/07/29/android-malware-steals-info-million-phone-owners/

[2] http://news.cnet.com/8301-27080_3-20013222-245.html

[3] http://www.bbc.co.uk/news/technology-10912376

[4] http://www.reuters.com/article/idUSTRE66T52O20100730

[5] http://us.mcafee.com/root/product.asp?productid=mobile_info

[6] http://www.symantec.com/norton/smartphone-security

are constrained by its smartphone host, a platform characterized by its simpler design and weaker computing capability compared with a desktop system. As an example, consider the permission of microphone access, which has to be granted to applications such as a voice dialer. The threat of a malware with such a permission can be mitigated by those existing approaches. Specifically, a behavior-based malware detector [3] can pick up anomalous behavior such as regular CPU-intensive operations and heavy use of bandwidth, which could be associated with activities like performing an in-depth speech recognition and transmitting a large amount of phone recordings (typically, on the order of 100 KB per minute) to the Internet. A reference monitor could deny installation of applications asking for both microphone access and other dangerous permissions: particularly, access to the numbers being called, which allows malware to target a small set of calls involving high-value information, and Internet connections. As a result, the malware is left without any apparent way to communicate stolen information to its master.

Contrary to this intuition, we show that sophisticated malware can be built over the smartphone platform to evade such defenses. This is possible because of two new observations. First, the context of a phone conversation can be predicted and fingerprinted under some circumstances, which enables an efficient analysis to extract a small amount of high-value information from the conversation. A prominent example is one's interaction with an automatic phone menu service, also known as interactive voice response (IVR) system, which is routinely provided by customer service departments of different organizations (e.g., credit-card companies). The detailed steps of such an interaction were found to be easily recognizable in our research, from a small set of features of the conversation and related side-channel information. As a result, sensitive data such as credit-card numbers can be accurately identified at a small cost. Second, like other computing systems, smartphones contain a set of built-in *covert channels*, which can be leveraged to transmit a small amount of sensitive information without direct access to the Internet. To demonstrate that this threat is realistic, we present an example of such malware in this paper, called *Soundcomber* , a sound Trojan that masquerades as an application with the legitimate need to use the microphone, such as a voice dialer or a voice memo application. Soundcomber is capable of stealing a user's credit-card number from her interactions with credit-card companies' IVR. This is achieved through a suite of techniques for *hotline detection*, *profile-based extraction*, *lightweight speech/tone recognition* and *covert-channel communication*. The hotline detection component analyzes the initial part of a call to determine whether an IVR is called, and if so, which IVR (based on IVR fingerprinting). Based on the detected IVR, Soundcomber uses a preset profile (state machine) for that IVR and intelligently analyzes a phone menu to determine the interaction path, i.e., the sequence of menu selections in terms of the digits a user enters, that leads to the situation where the user has to reveal her credit card number.

Although performing speech recognition over the whole recording is computationally intensive, Soundcomber only needs to work on a small portion of it, according to the profile, to identify the digits a user speaks or types to the IVR, which turns out to be lightweight. Of particular interest here is the analysis of typing: the tones produced thereby are actually not part of the phone conversation. We demonstrate, however, that they can be picked up by the microphone when the tones are played back to the user. While this is not surprising, it turns out to be technically challenging to extract information from this audio side channel, because the tones are drowned out by background noise in the recordings — the microphone picks up a faint "echo" of the digits pressed. Using tailored signal processing techniques we show that it is feasible to isolate these tones and recover the actual digits pressed with high accuracy. We note that even though we use credit card numbers as a proof of concept, the same technique can be applied to target other valuable information such as shorter PIN numbers, social security numbers (the last four digits are often requested as part of authentication), passphrases such as mother's maiden name, and so on. Thus, even though profile-based processing of text transcripts can be done offline, profile-based processing on the smartphone itself (a) reduces the amount of resources needed to process the entire speech recording to generate the transcript, (b) reduces the amount of data sent by the smartphone which would be noticeable if all recorded phone calls are uploaded and (c) relieves the burden of the malware master to process potentially lengthy transcripts from a large number of sources (in Section 4 we provide some conservative estimates to show such costs can be prohibitive).

Because Soundcomber is doing the processing and extraction of relevant data locally on the phone, the large amount of phone call recordings can be distilled into a very small amount of valuable data. If the whole recording were transmitted to the master, the data required to be transmitted would be several orders of magnitude larger. Further compounding this communication woe is the fact that the malware *cannot* access the number being called and therefore would have to record and transmit *every single* phone conversation if the processing was not done locally. The communication/computation overhead incurred thereby would significantly reduce the stealthiness of the malware. Given the much simpler task of transmitting merely 16 digits of a credit-card number, Soundcomber can easily make the communication less observable: for example, this can happen through a legitimate network-facing application, such

as a browser. In the presence of a colluding application with a networking permission, which we found is easy to find or install (see Sections 4.1 and 4.2), Soundcomber can pass the digits to it through a covert channel. This even evades the protection based on mediating the overt communication between applications, as described in a recent proposal [10]. Our research discovers multiple covert channels on the smartphone platform, including file locks, vibration and screen settings. Leveraging these channels to transmit the digits is found to be completely practical.

Finally, since no existing defenses work on Soundcomber, we designed and implemented a defensive architecture that foils the malware. In essence, all audio recording and phone call requests are mediated by a reference monitor, which can disable (blank out) the recording when necessary. The decision on when to turn off the switch is made according to the privacy policies that forbid audio recording for a set of user-specified phone numbers, such as those of credit-card companies. We evaluate our prototype defensive architecture and show that it can prevent our demonstrated attacks with minimal processing overhead.

We now summarize our major contributions:

- *Targeted, context-aware information discovery from sound recordings*. We demonstrate that smartphone-based malware can easily be made to be aware of the context of a phone conversation, which allows it to selectively collect high-value information. This is achieved through techniques we developed to profile the interactions with a phone menu, and recover digits either through a side-channel in a mobile phone or by recognizing speech. We also show how only limited permissions are needed and how Soundcomber can determine the destination number of the phone call through IVR fingerprinting.

- *Stealthy data transmission*. We studied various channels on the smartphone platform that can be used to bypass existing security controls, including data transmission via a legitimate network-facing application, which has not been mediated by the existing approaches, and different types of covert channels. We also discovered several new channels, such as vibration/volume settings, and demonstrated that covert channel information leaks are completely realistic on smartphones.

- *Implementation and evaluation*. We implemented Soundcomber on an Android phone and evaluated our technique using realistic phone conversation data. Our study shows that an individual's credit-card number can be reliably identified and stealthily disclosed. Therefore, the threat of such an attack is real.

- *Defensive architecture.* We discuss security measures that could be used to mitigate this threat, and in particular, we designed and implemented a defensive architecture that prevents any application from recording audio to certain phone numbers specified by privacy policies.

2 Overview

Assumptions. Soundcomber is designed to work under limited privileges. Specifically, we assume the Trojan is granted access to the microphone, as required by its legitimate functionality, but is denied network connections and other risky permissions. Simultaneous access to microphone and networking is well known to be a dangerous combination of permissions that should not be bestowed to untrusted code [5], as a user's speech is not supposed to be recorded and transmitted to untrusted recipients. The malware is also denied other risky permissions such as intercepting phone calls. It can acquire other information necessary for its mission, e.g., the phone number being called, through analyzing phone recordings. Avoiding dangerous permission combinations can be achieved during the installation of an application: as an example, Android explicitly displays the permissions requested by an application and asks the user whether to grant these permissions (although the options are limited to *install/do not install*). Alternatively, a system like Kirin [5] could be used to disallow dangerous combinations.

Architectural overview. *The main goal of Soundcomber is to extract a small amount of high-value private data from phone conversations and transmit it to a malicious party.* It also aims to do so in a *stealthy* manner, by evading detection and not degrading the user experience, and under possibly restricted configurations as described above. These goals are served by a design illustrated in Figure 1, which includes two key components: a *context-aware data collector* (*collector* for short) and a *data transmitter* (*transmitter*). The collector monitors the phone state and makes a short recording of the calls it deems interesting based on a *profile database*. The recording is then analyzed based on the specific profile to extract user data that is passed to the transmitter, which manages to send it to the malware's master. Since Soundcomber does not have direct access to the Internet, this transmission needs to be done through a second application, either a legitimate network-facing application like the browser or a colluding program with the networking permission. To deliver the data to the latter, the transmitter needs to use covert channels when overt communication is monitored by a protection mechanism [10]. In the following we explain how the Trojan can be used to steal a phone user's credit-card number.

Detailed credit-card theft scenario. Armed with access

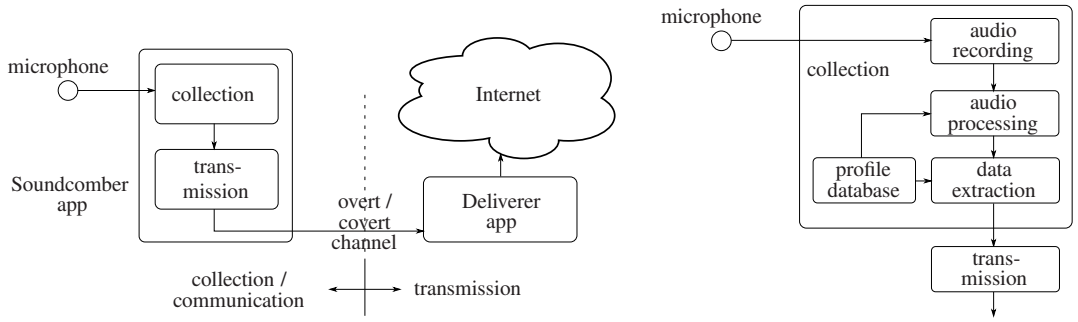

Figure 1. The left drawing shows the architecture of Soundcomber with the collection and communication part on the left, connected through an overt or covert channel to a second application on the right which can access the Internet and forward extracted data. The right drawing focuses on the collection part of Soundcomber. Audio is recorded using the microphone, and processed, and high-value data is extracted and forwarded to the communication part.

to the microphone, Soundcomber records a person's call and performs an audio analysis of the recording. The processing of the audio (i.e. recognizing speech and touch tones) and data extraction (i.e. extracting relevant information from transcribed speech/tones) is *profile-driven* so that speech/audio processing is targeted at specific types of information. As a proof of concept, we demonstrate Soundcomber's effectiveness at extracting credit-card numbers from spoken as well as touch-tone based audio samples. In this case, the profile contains a state machine of a credit-card company's IVR system, i.e., the automated menu-driven systems usually encountered when calling customer service, thereby allowing Soundcomber to understand the semantics of various parts of the audio recording through a very lightweight analysis and target specific regions of the audio for extracting the speaker's credit card number. An example of the profile is the sequence of the digits the user enters for selecting different menu options, which can be built through analyzing the IVR menu of a specific credit-card company. This sequence can be easily recognized by Soundcomber's collector component from the tones of individual digits. Such a profile-driven analysis provides a general approach to target specific regions of speech samples and extract precise and relevant information for improved analysis and minimal transmission requirements of such data. More specifically, the identified regions go through a tone/speech recognition, which is tuned to identifying digits and therefore very efficient. Once the credit card digits have been extracted, it is sent by the transmitter component to the adversary in one of several ways. In the case of a restricted security configuration, the transmitter has several available options: it can leverage an existing application such as the web browser by directly invoking it to load a URL to a malicious website, thereby transmitting sensitive information with relative ease; alternatively, it can use one of several covert channels on the Android platform, when a paired malicious application with network access is present. In Section 4.2.1, we discuss several ways to ensure such a paired application is installed.

In the follow-up sections, we elaborate our designs of the collector and the transmitter, and our implementation of Soundcomber on an Android phone. We also show that the threat posed by the malware can be mitigated using a context-sensitive reference monitor, which blocks audio recording when certain numbers are being called.

Video Demonstration. We have uploaded a video demonstration of Soundcomber that shows the entire process from calling a real credit-card company, to the (fake) credit card number being extracted through audio analysis, transferred to a paired Trojan application via a covert channel, and then to the (pretend) Malware master's server (located in another country). We point the reader to our video demo at `http://www.youtube.com/watch?v=_wDhzLuyR68`.

3 Context-Aware Information Collection

The collector is designed to monitor the phone state to identify and record phone conversations of interest, then decode the recording to perform a lightweight analysis, which uses tone/speech recognition and the profile of the call to locate and extract high-value information. This process is illustrated in the right part in Figure 1. Here we elaborate its design and implementation.

3.1 Audio recording

We now describe how Soundcomber can acquire an audio recording of a phone call along with the number that was

dialed, which is later used for profile-based data extraction.

When to record. The first step to extract high-value information such as credit-card numbers, is to record the user's phone conversation. To this end, Soundcomber monitors the phone state and starts recording whenever the user initiates a phone call. This step is performed in a completely unobtrusive and stealthy fashion — Soundcomber does not even have to be running prior to the phone conversation, as it will be started automatically by the Android OS.

Recording in the background. Once Soundcomber is invoked when a phone conversation is initiated, it starts recording the audio input from the microphone. This recording is done in the background and no indication is given to alert the user that the call is being recorded. Soundcomber stops recording when the call has ended or after a pre-defined maximum recording length. Since one's sensitive information, such as credit-card number, social security number, etc., is often required at the beginning of a phone conversation with an IVR, the recording can be short, typically a few minutes.

Determining the number called. After a call has ended, Soundcomber needs to decide whether the recording deserves analysis. Soundcomber makes this decision based on *profiles* specific to the number called. For example, if a credit-card customer service line is detected, Soundcomber knows that the recording could include a credit card number and therefore starts working on it using the profile of the service. While Android offers a special permission, *intercept outgoing phone calls*, that allows to easily determine the called number, this is deemed an unusually high privilege with significant security implications. A less obtrusive path to retrieve the called phone number is by going through a call list and extracting the number of the most recent call. The permission needed to access the call list is less sensitive and is shown as *read contact data* (it is conceivable that voice dialer and memo apps would benefit from access to the contact data). Nevertheless, to reduce the permissions necessary for Soundcomber we decided to determine the number called without any additional permissions by using the data already collected by Soundcomber, namely the audio recording.

By analyzing the beginning of a recording, Soundcomber checks whether it matches an internal database of service hotlines and if that is the case, the recording is processed further. Specifically, the analysis consists of looking at the first segment in the recording, which for a service hotline is typically a greeting or introduction. Soundcomber will run speech recognition on the first segment and compare the extracted words to an internal database of keywords for different hotlines. If a match is detected, the recording is processed further as described in Section 3.2, using the profile of the detected hotline. To pick the defining keywords for each hotline, we wrote a program which analyzes several samples of hotlines and determines the relevant, non-overlapping keywords for each hotline. As an example, the hotline of HSBC in America greets a user with "Thank you for calling HSBC, the world's local bank." Our tool determined that the keywords which were recognized consistently and which did not overlap with other hotlines were *for*, *calling*, *local* and *banking*. In fact, *banking* is not contained in the recording, but the speech recognition consistently returned *banking* as one of the keywords (in fact "misrecognizing" *bank*), and as such it became a relevant and reliable keyword.

The advantage of analyzing the beginning of a recording is that the permissions necessary for Soundcomber can be kept to a minimum. In return, Soundcomber has to spend more time analyzing recordings to detect service hotlines. Using either of the other two methods above (intercept call or go through the call list) would be less expensive in terms of computation required and would also be more accurate, but would require Soundcomber to declare an additional permission (*intercept outgoing phone calls* or *read contact data*) potentially looking suspicious. In summary, with limited permissions to 1) *read phone state* and 2) *record audio*, Soundcomber can record outgoing phone calls. Working under these permissions, Soundcomber does not know the number of an ongoing call, and thus needs to make and analyze a short (e.g., $<$ 1 minute) recording of every call and then discard the recording if no corresponding profile is found.

3.2 Audio processing

We now describe the second stage of the collection process. If a profile for the dialed number is found, Soundcomber proceeds with *audio processing* (shown in Figure 2) to first decode the sound file, then determines whether speech or tone recognition is needed using lightweight analysis, and finally proceeds with either speech or tone extraction aided with profiles for targeting sensitive information. We discuss profiling in detail in Section 3.3.

3.2.1 Tone recognition

The traditional analog telephone system uses tones to allow users to navigate the IVRs frequently used by customer service lines. Specifically, today's systems use dual-tone multi-frequency (DTMF) [6] to transmit keypad presses on a phone or mobile phone. For each possible key (0 to 9, *, #) a combination of two tones is sent in-band through the voice channel (see Table 1). To send a "4", for example, the phone will generate two sine tones, one at 770 Hz and one at 1209 Hz and send them through the voice channel.

Technically, mobile phones do not send actual tones but instead use a signaling channel to inform the mobile phone

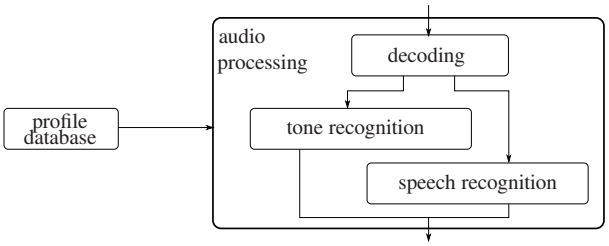

Figure 2. This figure shows the audio processing step of Soundcomber, with the decoding of the audio and tone and speech recognition before the output is forwarded to the data extraction (not visible).

Table 1. DTMF tones

	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

network of the pressed key. Nevertheless, a mobile phone usually plays the corresponding tones locally to give aural feedback to the user. We discovered that such tones are leaked through a side-channel and can be used to identify the phone user's inputs. Specifically, although the tone is played back only in the earpiece of the phone, the sound is conducted through the inside of the phone and picked up (faintly) by the microphone. Our experiments confirmed that the tones can still be detected accurately, precisely because they occupy specific frequencies, and can be reliably identified even with low signal-to-noise ratios.

For Soundcomber we used Goertzel's algorithm [12], an algorithm often applied for recognizing DTMF tones in digital signal processing [11]. Goertzel's algorithm is more accurate and also more efficient than a general FFT when used to detect DTMF tones. This helps cover Soundcomber's analysis operations by using less processing power. Nevertheless, using an unmodified Goertzel still did not provide the necessary accuracy. The tone energy levels are much lower than what is normally the case because they are recorded through a side channel. To increase noise rejection, we developed *adaptive and frequency-dependent* detection thresholds as described below.

Using the Goertzel algorithm, Soundcomber calculates the spectrum for all relevant frequencies f_i. The algorithm processes individual frames consisting of N consecutive samples and outputs i spectral coefficients for each frame. Consider an audio recording containing n frames with N samples each. We compute the average spectral power for each frequency i over the whole recording as follows:

$$\gamma_i = \frac{1}{n}\sum_{k=0}^{n-1} \mathcal{G}_N(x(kN,(k+1)N-1)),$$

where $\mathcal{G}_N$ is the Goertzel algorithm using N samples (corresponding to one frame). For each frequency we set a threshold $\gamma_{thr,i} = \beta \cdot \gamma_i$ to detect in which frames frequency i has a peak, with β constant across all i. Applying different thresholds for different frequencies removes detection errors caused by frequency-dependent noise in the recordings.

Soundcomber processes the audio recording frame by frame and for each frame and for each frequency compares the spectral energy of a particular frequency j, γ_j, with the frequency-specific threshold $\gamma_{thr,j}$: if $\gamma_j > \gamma_{thr,j}$ the frequency j is assumed to be present in the current frame. In our prototype we used a value of $\beta = 2$ so the thresholds used were twice the average signal energy per frequency. Those thresholds were found to be effective at identifying tone signals (see Section 6).

After detecting the peaks, Soundcomber checks each frame to see whether the peaks exceed the thresholds. Ideally, if a DTMF tone is present in the recording, exactly two frequencies are detected and the corresponding tone can be determined. Noise in the recording can lead to the detection of more than two frequencies. When this happens, all detected frequencies except the two with the highest peaks are eliminated. In the following, we present a visual example of this detection process. In the example, three peaks are detected (corresponding to the 1's in the peak vector). The lowest peak is eliminated to yield two peaks, which correspond to the DTMF tone for "4".

$$\left.\begin{array}{l}\gamma_1 \overset{?}{>} \gamma_{thr,1}\\ \vdots\\ \gamma_i \overset{?}{>} \gamma_{thr,i}\end{array}\right\} \rightarrow \text{peak detection} \rightarrow \begin{pmatrix}1\\0\\\vdots\\1\\1\end{pmatrix} \rightarrow$$

$$\text{peak elimination} \rightarrow \begin{pmatrix}1\\0\\\vdots\\1\\0\end{pmatrix} \rightarrow \text{tone detection} \rightarrow \text{“4”}$$

Moreover, a tone is deemed to be identified only after the combination of its two frequencies has been found consistently in multiple frames in a row, typically at least 2 frames.

3.2.2 Speech recognition

To perform the speech recognition part for Soundcomber we first looked at the available options. The Android platform contains speech recognition functionality using a Google service, but the functionality is not usable for Soundcomber

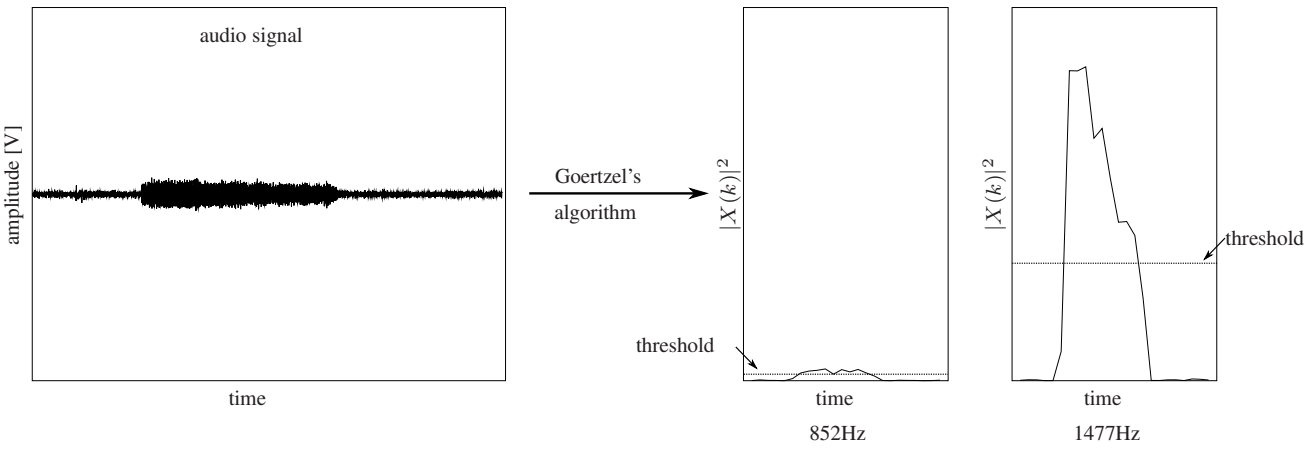

Figure 3. The audio signal shown in the left part is analyzed using Goertzel's algorithm that generates the two spectra on the right (for two different frequencies). Each frequency has a custom threshold (dotted line). In this example, the spectral energy of both frequencies is above the threshold simultaneously for several frames in a row, indicating that the digit "9" was pressed.

for several reasons. First, Google's speech processing cannot detect tones for tone recognition. Second, it can only do speech recognition interactively by prompting the user, and cannot be run in the background on an already recorded audio file.[7] In addition, the voice application offloads the processing to Google by uploading the recorded audio over the Internet, which is easily noticeable and thus not suited for Soundcomber.

We instead ported PocketSphinx developed by the Carnegie Mellon University[8] to Android using the Native Development Kit (NDK) and wrote an intermediate layer between PocketSphinx and Soundcomber and integrated both parts using the Java Native Interface (JNI).

Soundcomber segments the audio (described below) before processing the individual segments in PocketSphinx to get the transcribed text. Segmentation of audio allows Soundcomber to work only on some portions of the audio, which reduces the overhead of speech recognition, thereby making Soundcomber stealthier. Currently, Soundcomber focuses on stealing information that can be transmitted as digits (e.g., credit card numbers, social security number, personal identification number, and so on) and we therefore adopted a language model that covers spoken digits for extracting information. The hotline detection described in Section 3.1 on the other hand uses a small language model for general spoken text.

Segmentation. Speech recognition with PocketSphinx is still fairly expensive in terms of processing time. To reduce this overhead to a minimum, Soundcomber does not work on the whole audio recording directly: instead, it preprocesses the recording to identify the segments that contain speech. Soundcomber first analyzes the signal power of the recording by calculating the mean power of individual frames containing n_s samples. For each frame k, we compute:

$$\delta_k = \frac{1}{n_s} \sum_{j=0}^{n_s} x(j)^2$$

The average power over the whole recording with n_f frames is then computed as:

$$\delta_{\text{Recording}} = \frac{1}{n_f} \sum_{k=0}^{n_f} \delta_k$$

By comparing the signal power δ_k of individual frames with a threshold $\delta_{thr} = \alpha \cdot \delta_{Recording}$ for a certain α, we can locate the audio segments with sufficiently high signal power levels, which indicates the presence of relevant sounds:

$$\text{if} \quad \begin{cases} \delta_k \leq \delta_{thr} & \text{silence} \\ \delta_k > \delta_{thr} & \text{sound} \end{cases}$$

Like tone recognition, we also check the presence of multiple consecutive frames with signal power levels above the threshold, which collectively form a segment. This simple approach is both efficient and effective, as shown in our experimental study. Assuming for example that only 20% of a recording contains speech (as could be the case when navigating an IVR), then 60 seconds of audio would take

[7] http://www.4feets.com/2009/04/speech-recognition-in-android-sdk-15/

[8] http://www.speech.cs.cmu.edu/pocketsphinx/

about 21.4 seconds of processing time if speech recognition is run on the whole recording, but only 5.2 seconds if the audio is segmented first and only the segments with actual speech are run through the speech recognition.

3.3 Targeted data extraction using profiles

The objective of Soundcomber is to extract a small amount of high-value data from an audio recording. This cannot be achieved without understanding the semantics of the recorded audio. Although an effective semantic analysis in general can be hard, our research shows that it can certainly be done for some specific scenarios, particularly interactions with IVRs. Such an analysis is based upon a predetermined profile, which indicates to Soundcomber how a sequence of user behaviors (e.g., the digits entered or spoken) can move an IVR to the state where sensitive user data is input. The profile also allows Soundcomber to skip over the segments that do not involve useful information, thereby reducing overhead.

In their general form, profiles model a context and describe the location of high-value information under such a context. An example of the context can be audio features of some keywords, like "enter your PIN", which is supposed to be followed by one's password. Again, our research focuses on IVRs, whose interactions with a phone user can be modeled as simple state machines. The sequence of such transitions, as observed from the user's digit inputs recovered from an audio recording, points to the position where credit-card numbers or other important data is entered.

We first describe the profile for a phone menu system and then discuss how to apply the profiling idea more generally.

3.3.1 Profiling phone menus

Many businesses and organizations offer hotline services through IVR systems. An IVR includes a phone menu, which guides the caller step by step to the service she is seeking. During such a process, confidential user information, such as credit-card number, social security number, PIN, etc., is input by the user for authentication and other purposes. As an example, the phone menu of Chase bank lets a customer first press "2" on the main menu, then "2" on the submenu and finally "1" before asking a customer to enter their 16-digit credit-card number. Therefore, a profile for this menu needs to include a sequence ('2', '2', '1', CC number), indicating the expected input to reach a state in the IVR with high-value information. Following this sequence, Soundcomber can easily locate the segment that includes a credit-card number. The same idea can be applied to model the interactions with other IVR systems. The adversary can analyze an IVR phone menu offline, identify all the sequences that lead to the desired user input, and then assemble these sequences into a finite state machine, which serves as the profile. Figure 4 shows the state machine corresponding to the IVR described above. In our current proof-of-concept implementation, Soundcomber stores linear sequences and detects credit numbers after the sequence "2, 2, 1" for example. A profile can be expressed as $\{d_1, d_2, d_3, \{\text{target}\}\}$, where the d_i are the expected sequential inputs and $\{\text{target}\}$ the information which will be extracted. We leave a full implementation with general finite state machines to future work.

To use such a profile, our prototype of Soundcomber first runs tone recognition on the recorded audio to recover the digits and lightweight speech recognition on the identified segments to extract spoken digits. The output of tone and speech recognition is combined into a transcript, which is used to explore the state machine and identify the high-value information in the transcript. If at any point a state is reached which does not lead to high-value information, the analysis is stopped and the recording discarded. A further optimization is to run only tone recognition first and use the digits to explore the state machine. Once a state is reached where high-value data is available, the speech recognition is run on the relevant segments.

3.3.2 General profiles

Apart from the finite state machine-based profiles, the context of phone conversations can also be fingerprinted in other, more generic ways, as described below. In-depth studies and implementation of these approaches are left to future research.

Speech signatures: Soundcomber could take advantage of the sound samples included in the incoming audio such as the voice prompt to "speak or type your credit card number now". If this phrase is detected, then the outgoing audio immediately after this signature is likely to be a spoken or typed credit card number.

Sequence detection. Since a sequence of 12–16 digits is quite likely a credit card number, profiles can also instruct Soundcomber to scan for long numeric sequences. While this method will work in some cases, we found that a semantic understanding of the speech transcript aided in picking out specific, targeted pieces of information. Nevertheless, sophisticated combinations of different techniques can improve the accuracy of detection.

Speech characteristics. Profiles could specify certain sound features that are typically exhibited by spoken credit card numbers. For example, Soundcomber could perform a high-level analysis of speech to hone in on specific features unique to spoken credit card numbers (e.g., rhythmic or monotonous speech). Once these features are observed, Soundcomber can apply targeted speech extraction on the

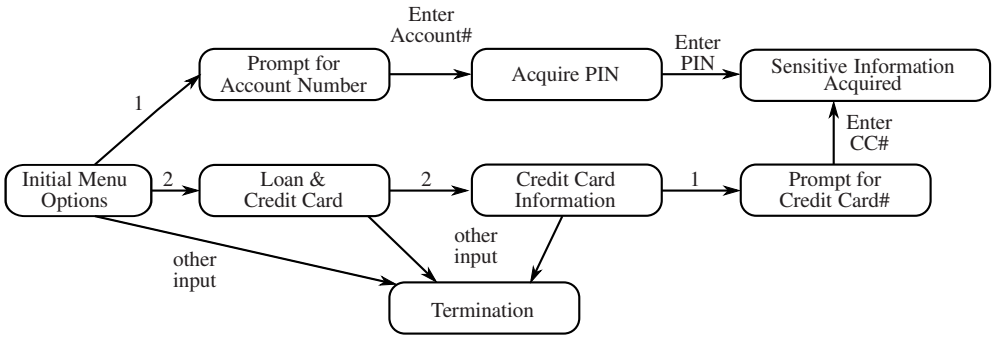

Figure 4. A model of a service line with two different paths leading to high-value information. The branches indicate the input required by the user to take the corresponding transition. For this IVR, the expected input sequence for reaching high-value information would either be "1" or "2, 2, 1".

identified segments.

4 Stealthy Data Transmission

Once the sensitive information has been obtained via the collection phase, it must be transmitted to the malware's master stealthily. This is where the advantage of Soundcomber of processing recorded audio and extracting high-value information locally becomes apparent. If the malware master were to do processing centrally, Soundcomber would have to transmit approximately 94KB of data for each minute of recorded audio. This would not only make it more difficult for Soundcomber to operate stealthily, but would also place a much larger burden on the infrastructure of the malware master. Assuming for example anywhere between $100,000$ and $1,000,000$ infected smartphones[9], this could easily generate in the order of several tens of gigabytes to a terabyte of data per day[10] which would need to be processed centrally by the malware master. By extracting the relevant information locally, Soundcomber significantly reduces the data necessary to be transmitted to the malware master. For one call, a one-minute recording would require 94KB to be transmitted to the malware master if no local processing is done, but if a credit card number is extracted locally then at most 16 bytes have to be transmitted, a reduction by a factor of 6000. Transmitting such a small amount of data is much easier done stealthily and the remaining work for the malware master becomes minimal.

As already mentioned, we assume Soundcomber does not have permission to use the network to circumvent tools such as Kirin [5]. In fact, a recent paper [10] further proposes to mediate the explicit communication between two untrusted applications, by restricting access from one application to another based on installation-time permissions and run-time state. We present two methods that will circumvent such prevention and detection mechanisms. In the first method, Soundcomber uses a legitimate, existing application with network access (such as the browser) to transmit the sensitive information. In the second method, Soundcomber uses a paired Trojan application with network access and communicates with it through a covert channel. Both methods circumvent known, existing defenses.

4.1 Leveraging third-party applications

The permission mechanism in Android only restricts individual applications, not the relations between applications. This allows Soundcomber to communicate with its master through a legitimate network-facing application, such as a web browser. Specifically, the malware can request the browser to open an URL in the form `http://target?number=N` with N the credit card number to pass it to a target web site. A weakness of this approach is that the transmission is more noticeable to the user because the browser will be brought to the foreground. Such an activity, however, could be easily "explained away". For example, an application which displays ads in its interface can pretend to open a browser window for a more detailed version of an ad, first transmitting the valuable information before immediately redirecting to the actual ad. Or a user could be tricked into believing that a new browser window is opend caused by a stray click that leads to a standard sites such as Google or CNN. Nevertheless, we consider this approach to be more intrusive than a paired Trojan application, which once installed, performs all such communication in the background.

[9] Android malware steals info from one million phone owners—`http://www.sophos.com/blogs/gc/g/2010/07/29/android-malware-steals-info-million-phone-owners/`.

[10] Assuming between $100,000$ and $1,000,000$ infected smartphones, on average 5 calls a day, where Soundcomber records the first minute of each call.

4.2 Covert channels with paired Trojans

Next we consider communication between two Trojan applications. In this case, Soundcomber is paired with a *Deliverer* Trojan with network access, which transmits the extracted sensitive information (typically only a few dozen bytes) to the malware master over the Internet. Under the current Android security model, the Soundcomber and Deliverer applications could communicate through overt channels, however such communication will be limited with recently proposed defenses [10]. To be as stealthy as possible and to circumvent such defenses, covert channels on the Android platform can be used instead to covertly transfer the extracted information from Soundcomber to Deliverer and thereby to the malware's master. This paper thus also identifies and evaluates *new* covert channels of communication on smartphone platforms and demonstrates that communication through such channels is realistic for sensory malware.

4.2.1 Installation of paired Trojan applications

To leverage a second installed application (Deliverer) for transmitting the extracted information to the Internet, we have to convince the user to install such an application. We assume that Soundcomber itself is packaged as a Trojan into an application that is attractive enough to get users to install it. Once the user has installed Soundcomber itself, we have to make sure that the Deliverer application is also installed. We have explored a number of options and believe that they will trick enough users into installing the Deliverer application.

Since an Android application can launch the installation of another application, we investigated two possibilities:

Pop-up ad. The Deliverer application could implement a cover functionality and hide the transmission functionality as a Trojan part. When the user first executes the cover functionality of Soundcomber, a pop-up ad is displayed, advertising the Deliverer application (respectively its cover function) as another fun, hip, cool, useful application. When clicking on the ad, Soundcomber will tell Android to open a web browser with the webpage of the Deliverer application or even directly initiate the download in the web browser.

Packaged app. An Android application can include custom resources, e.g., the installation file for the Deliverer application can be included in the Soundcomber app. When Soundcomber is started for the first time, it launches the installation of the included application package for the Deliverer app. Users might thus be tricked into confirming the installation, or be gently persuaded by informing them that a "necessary helper application" has to be installed before Soundcomber (respectively its cover function) can be used.

4.2.2 Covert channels on the smartphone

We discovered several covert channels on the Android platform, some of which are specific to Android, and others that are specific to Android's underlying Linux system. The channels have different properties in terms of stealthiness, transmission rate and reliability (i.e., error rate). For Soundcomber even low bit-rate covert channels are sufficient to transmit the high-value information, which is typically very small. Here we describe a few covert channels we found on Android that make use of vibration settings, volume settings, the handset screen, and file locks.

Vibration Settings. The covert channel based on *vibration settings* is specific to Android. Any application can change the vibration settings and every time this setting is changed the system sends a notification to interested applications. Our prototype exploits these notifications as a communication channel. Soundcomber codes the sensitive data into a sequence of vibration settings and then applies these settings sequentially. Deliverer listens to the setting changes and decodes them back into the data.

While changing the settings might indirectly be noticed by the user if a call or message arrives during transmission, saving and restoring original settings after the transmission as well as transmitting at opportune times (e.g., at night) can mitigate this problem.

In our experiments we achieved a bandwidth of 87 bits per second through this channel (see Section 6). Higher bandwidth was prone to overload the Android system with notifications. Nevertheless, 87 bits seems sufficient for transmitting the small amount of data Soundcomber collects: for example, sending the 16 digits of a credit-card number (54 bits) takes less than 1 second. Other advantages of using this channel include that *no permissions* are needed to access it and it does not leave any traces. We thus suggest that isolation mechanisms like the one proposed in [10] for smartphones must also restrict covert communication through event notifications, otherwise they will not provide *complete mediation* of communication channels.

Volume settings The *volume-setting covert channel* is similar to the vibration-setting channel. The difference is that changes in the volume setting are not automatically broadcasted, which means that two applications communicating through this channel have to set and check the volume alternatively, requiring tighter synchronization in time (i.e., the receiving application has to be certain that it only checks after next setting has been set by the sending application).

On the other hand, because the volume is a setting between 0 and 7, the channel allows Soundcomber to transmit 3 bits per iteration. Generally speaking, the sending application will set the volume at times $t_s = k \cdot t_i$ ms within each second, for $k = 0, \ldots, (\frac{1000ms}{t_i} - 1)$ while the receiving application will read the volume setting at times

$t_r = k \cdot t_i + (t_i/2)$. Initial experiments indicated that setting and getting the volume settings takes on the order of 7ms, so we set the iteration length to $t_i = 20$ ms and thus achieved 150 bps. Specifically, the sender will set the volume at times 0ms, 20ms, 40ms, and so forth, while the receiver will read the volume at times 10ms, 30ms, 40ms, and so on. At this speed, however, here is a small chance that if Android is doing some "housekeeping" during the transmission that either the sender or receiver will miss a window and get out of sync. No permission is needed to exploit this channel.

Screen Another channel specific to Android, which might well be specific to mobile phones, is the mobile phone screen. This channel is particularly interesting because it turns out to be an *invisible visible channel*. Mobile phones typically conserve power by switching off the screen if the user is not using it for a certain period of time. The screen can be re-awakened through a press of a button or touching the screen.

Android allows individual applications to influence the screen. For example, applications that need to prevent the screen from dimming out (e.g., a car GPS application) can request a *wake-lock* from the operating system. Acquisition of the wake-lock immediately turns on the screen if it was dark, and the screen will only be switched off again when the lock is released. The change of the screen states (on or off) triggers a notification mechanism, which Android uses to inform applications of the screen setting. These notifications are used by Soundcomber to create a covert channel with its colluding application. More specifically, Soundcomber acquires and releases a wake-lock at regular intervals, which transmits to Deliverer one bit of information for each iteration through the notifications issued by the system.

At first glance, this channel is not stealthy: the alteration of screen states seems to be too conspicuous to go unnoticed. We found, however, that on the Android G1 phone, if the wake-lock was held for a short enough time, a latency in the electronics of the device would prevent the screen from actually turning on, but the notification that the screen had been turned on was still sent. The channel is thus again invisible to the user.

Compared with the vibration and volume settings channels, exploiting this channel needs the permission `WAKE_LOCK`, which is explained by Android as *prevent phone from sleeping*. Also, the bandwidth of the channel is fairly low, less than 5.29 bits per second. Nevertheless, it offers a practical way to deliver a small amount of data within a short period of time: for example, sending a 16-digit credit card number (54 bits) takes around 11 seconds.

File Locks Covert channels using file locks have been known since 1989 [9] and are far from specific to the Android platform. We show, however, that this channel also works well (we characterize the bit rate) on a smartphone platform and can be used to practically leak sensitive data to an unauthorized party. The basic idea is that two applications can stealthily exchange information through competing for a file lock.

Specifically, if Soundcomber wants to signal a 1 to Deliverer, it requests a file lock on a file shared between them. Deliverer also tries to lock that file: if this attempt fails, '1' is sent, otherwise, '0' is sent. Communication through this channel can even evade the stringent Bell-LaPadula (BLP) [2] model: a high process (the process with access to microphone) can read-lock the shared file, while the low process (Deliverer) tries to get a write-lock on it. This does not violate the "no-reads-up" and "no-writes-down" policies of the BLP.

Our implementation of the file-locking channel on Android employs an efficient synchronization mechanism. Soundcomber and Deliverer each maintain m *signaling files*, $S_1 \cdots S_m$ and one *data file*. The signaling files are organized in a round-robin fashion. Before transmitting data, both parties lock their own signaling files (S_1 to $S_{m/2}$ for Soundcomber, $S_{m/2+1}$ to S_m for Deliverer) and Deliverer also blocks itself by attempting to lock S_1, a file already locked by Soundcomber. Soundcomber then sets/releases the lock on the data file according to the first bit it wants to transmit, and after that, wakes up Deliverer by releasing its lock on S_1 and blocks itself by waiting for the lock on $S_{m/2+1}$, the first of the signaling files already held by Deliverer. Deliverer then tests the data file to acquire the bit, removes its lock on $S_{m/2+1}$ to invoke Soundcomber and waits for the next bit by attempting to lock S_2. By rotating through the m signaling files, the two processes can synchronize themselves during the data transmission, which helps achieve a bandwidth of more than 685 bps. Developing the idea of rotating file locks became necessary because experiments showed that Android does not honor the sequence of file locking requests if the requests are made very closely spaced.

5 Defense Architecture

The current Android platform performs only static permission checks, without taking context information into account. For example, once an application is granted the permission to record the microphone, it can always make use of the permission, independent of the phone state (e.g., idle/call in progress) or the number currently called. This threat cannot be mitigated by the reference monitor architectures proposed in prior research [10]. Even though such work considers context while regulating inter-application communication, the collection of one type of sensor data based on other context information is not yet supported. In our research, we built the first defense architecture to

counter this threat. Our approach is not meant to be a replacement for the defense mechanism proposed by prior research. Instead, we intend to develop a new technique that can be incorporated into existing mechanisms. To this end, we implement a prototype to add a context-sensitive reference monitor to control the AudioFlinger service, the Android kernel service in charge of media data. This approach prevents audio data from leaking to untrusted applications during a sensitive call.

Our *reference monitor* is designed to block all applications from accessing the audio data when a sensitive call is in progress. It consists of two components:

- *Reference Service:* The *reference service* determines whether the phone enters or leaves a sensitive state by monitoring call activity. When a sensitive call is made the reference service alerts the *controller*. In our prototype the reference service is implemented in the RIL, the "radio interface layer" which mediates access from the Android OS to the baseband hardware. Any attempt to make a call, no matter how it is made, has to pass through the RIL. The reference service intercepts attempts to make outgoing calls and checks the called number. Whenever a call to a sensitive number is made, it notifies the *controller*.

- *Controller:* The *controller* embedded in the AudioFlinger service mediates access to audio data. It operates in one of the following two modes:

  - *Exclusive Mode:* In *exclusive mode*, the controller blanks all audio data being delivered to applications requesting audio data. Instead of the actual audio data, these applications will simply record silence.

  - *Non-Exclusive Mode:* In *non-exclusive mode*, the controller does not intervene and the audio data is delivered normally to applications.

When the reference service detects that a sensitive call is being made, it alerts the controller. On receiving the alert from the reference service, the controller enters exclusive mode and blanks all audio data being delivered to applications. Once the sensitive call has ended, the reference service again notifies the controller, which reverts back to non-exclusive mode. Our reference service can be used by existing reference monitor architectures to intercept phone calls, and use the controller to enable/disable recording from the microphone. Although we focus on audio data, the principle of adding context information to protect Android kernel services can be extended to protect other sensor data. We believe that existing architectures can use a similar technique to defend against sensory malware.

With the controller being a part of the AudioFlinger, we assume that the integrity of the Android OS itself is guaranteed. If the OS has been compromised by malware then the malware already has access to all data and can circumvent the controller. With the integrity intact, on the other hand, the controller can guarantee that no application can record audio while a sensitive call is in progress. Since the reference service is on the critical path to making a phone call, we measure the delay added by the service in Section 6 and show that it is negligible.

6 Evaluation

In this section, we report on our evaluation of Soundcomber. Our experimental study was aimed at understanding Soundcomber's capabilities to detect whether a hotline number was called, extract high-value data from a phone conversation using profiles, recover digits from tones/speech and transmit them through covert channels. We also wanted to determine the performance of these operations and the overhead they incurred.

6.1 Experiment settings

We carefully designed a set of experiments which studied hotline detection, tone/speech recognition, profile-based data discovery and different covert channels. Speech recordings from three participants were used.[11] Each participant was asked to speak or dial menu choices or credit card numbers, just as they normally would during a call to a bank service line. These credit card numbers were obtained from an online automatic generator.[12] After each phone conversation, Soundcomber analyzed the recordings, identified and delivered the credit-card numbers, which was monitored and measured to evaluate the effectiveness and performance of its operations. All the experiments were performed on an Android development phone with Firmware version 1.6 and kernel version 2.6.29-00479-g3c7df37. The phone contained a 1GB SD card and was connected to the Internet through Wi-Fi. We elaborate on the settings for individual experiments below.

Service hotline detection. When detecting whether the user called a service hotline it is important to minimize false positives. Too many false positives means that Soundcomber spends time analyzing phone calls that do not contain relevant sensitive information such as credit card numbers. We tested Soundcomber with 5 different service hotlines of financial institutions. For each hotline we recorded 4 samples and then extracted keywords using speech recognition to build a database of hotlines. The accuracy of the

[11] Indiana University IRB Approved Protocol ID: 1001000932.

[12] `http://mediakey.dk/~cc/wp-dyn/credit-card-number-generator.php`

detection was then tested using another set of 4 recordings each per hotline. To determine the false positive rate, we created 20 simulated normal phone calls by formatting normal speech from a corpus[13] to look like a phone conversation. We then ran the hotline detection on those simulated calls.

Tone recognition. To test the accuracy and performance of tone recognition, we recorded 20 samples of phone conversations with a phone line that we controlled.[14] The outcomes of the recognition were compared with the real digits the participant entered to determine the accuracy. We also measured the performance. The Goertzel's algorithm we implemented (see Section 3.2.1) utilized a frame length of $N = 205$ samples, corresponding to $25.625ms$. Our experiment required 5 frames in a row to minimize false positives when detecting DTMF tones, and used a detection threshold parameter $\beta = 2$ (see Section 3.2.1).

Speech recognition. The performance of speech recognition was tested by analyzing 60 recordings of simulated phone calls (20 samples from three subjects each) where a user spoke a credit card number by pronouncing individual digits. Again, the accuracy can be determined by comparing the recognized numbers with the spoken numbers and the performance was measured using the `getrusage()` function call.

Profile-based data discovery. To test the effectiveness of using profiles to discover high-value data, we created two profiles describing service hotlines. Participants then simulated 20 calls following a specific script for each of the hotlines and tested whether Soundcomber correctly recognized and extracted the high-value information. We also let the participants deviate from a given script to understand whether Soundcomber could correctly identify the operations that did not lead to high-value information.

Covert channel study. The most important performance measurement for covert channels is bandwidth in bits per second, which determines how long it takes to covertly transmit the extracted high-value information from Soundcomber to the Deliverer application. To measure the bandwidth of each channel, we ran a pair of applications to exchange 440-bit (55-byte) messages through the channel. All channels were parameterized to guarantee zero bit errors, sometimes at the cost of achievable bandwidth.

Reference monitor. To measure the performance impact of the reference monitor we first made the relevant modifications to the AudioFlinger service and then compiled the modified Android OS and installed it onto an Android developer phone (HTC Dream). We then made test calls to numbers marked as sensitive to measure effectiveness and performance.

[13] VoxForge—`http://www.voxforge.org/`

[14] Our demo shows a real phone call to a real credit card company, but we hesitated to have our subjects call real credit card company hotlines repeatedly. We thus had subjects follow a script for a simulated phone call.

6.2 Experiment results

6.2.1 Effectiveness

Service hotline detection. When testing the hotline detection on 20 recordings of actual hotlines, Soundcomber correctly identified 55% of the hotlines (among 5 different hotlines), detected 5% as the wrong hotline and missed 40% of hotline calls. Running the detection on 20 recordings of simulated normal conversations resulted in 100% being correctly identified as normal conversations, i.e., the false positive rate of the hotline detection is 0%. If a malware master includes detection information for the 5 largest financial institutions, the accuracy of the hotline detection is sufficient for Soundcomber to detect more than half of all the calls to those hotlines on average and analyze them, while not analyzing any calls containing a normal conversation. We deem this performance as sufficient for a malware master to collect a large number of credit card numbers.

Tone/speech recognition. Table 2 presents the accuracy of tone/speech recognition Soundcomber achieved. For speech recognition, Soundcomber identified 55% of the credit card numbers we tested without any error, and 20% with either one digit wrong or one missing digit. Note that single digit errors are often easy to correct: given the known digit pattern of credit card numbers and the use of the Luhn algorithm to remove invalid sequences, the brute-force search space is only 16 possible credit card numbers, or, when knowing the bank name, 12. The attacker could try charging each of these 12 numbers to see which one is valid. Tone recognition was found to be even more successful: Soundcomber could recover 85% of all credit card numbers correctly, and incur one-digit errors for the remaining 15%.

Detection by anti-virus applications. We tested two anti-virus applications for the Android platform: VirusGuard from SMobile Systems[15] and AntiVirus from Droid Security.[16] Neither of them reported Soundcomber as malware, even when it was recording audio and uploading data to the malware master.

Reference monitor. To test the performance overhead of both *reference service* and *controller*, we implemented them on an Android Developer Phone (HTC Dream). Since the reference monitor is on the critical path it is effective at blocking recording of audio from sensitive calls. We tested

[15] `http://www.smobilesystems.com/`

[16] `http://www.droidsecurity.com/`

Table 2. Accuracy of speech and tone recognition

	No error	1 error	2 error	>2 error
Speech	55%	12.5%	10%	5%
Tone	85%	5%	0	0
	1 missing	2 missing	>2 missing	
Speech	7.5%	7.5%	2.5%	
Tone	10%	0	0	

this functionality and present the performance overhead below.

6.2.2 Performance

Service hotline detection. As described earlier, Soundcomber looks at the first segment of a recording to determine the hotline. In our experiments the first segment had an average length of 6.1 seconds ($\sigma = 3.9$s) and recognition of the hotline took on average 34.6 seconds ($\sigma = 23.0s$). In general, hotline detection requires computation time of around 6 times the length of the segment analyzed.

Tone/speech recognition. The performance of tone/speech recognition is given in table 3, including time, power and memory. The recordings with speech inputs had an average length of 19.7 seconds, with a standard deviation of 4.485 seconds. It took Soundcomber 6.749 seconds on average to analyze and extract relevant menu choices and credit card numbers. However, due to other delays caused by I/O operation and scheduling, it took Soundcomber a total of 7.168 seconds on average from starting to output the final results. The recordings with tone-based inputs were much longer, 45.3 seconds on average. Presumably, this was caused by the extra time needed to switch between listening and tapping numbers. However, Soundcomber turned out to be effective at processing such recordings. It located credit-card numbers and extracted them within 5.524 seconds on average. The total time for such operations was found to be 7.694 seconds on average.

We ran PowerTutor[17] to measure the power consumption of Soundcomber. The average consumptions for analyzing speech and tone data were 94 mW and 101 mW respectively, which is higher than operations such as web browsing (varying between 54 mW and 87 mW, depending on the content of a web page). This is understandable because Soundcomber requires a number of computation-intensive operations. However, such difference does not seem to be significant. Also, Soundcomber could take measures to make its power consumptions less conspicuous, through distributing the analysis over a longer period time as discussed in Section 7. The memory usage was measured in our study by running the Android phone in debugging mode and using the Android developing plug-in for Eclipse[18] to read the heap size. Soundcomber took less than 3MB memory, which is reasonable, given some Android applications consume similar amounts of memory (e.g., vRecorder, voicememo, calendar and alarm clock: 2.8 MB each, VoiceDialer: 3.0 MB), and others could use even more (browser: 5.0 MB).

Covert channels. We also measured the bandwidths of different covert channels: the file-locking channel achieved 685 bps and the vibration channel transmitted data at a rate of about 87 bps. These two channels can deliver a credit card number in sub-seconds. The screen-setting channel was found to be much slower at 5.29 bps. Nevertheless, it was still able to transmit the 16 digits in 11 seconds. The volume channel was found to be 150 bps. This research shows that using covert-channels to leak sensitive information is completely practical on smartphones.

Reference monitor. Since the reference service resides in the RIL, it causes a certain delay when making a call. For a sensitive call, the reference service makes an RPC call to notify the controller, which on average causes a delay of 4.27ms. When a non-sensitive call is placed, no RPC call is needed and the time spent in the reference service is just 0.09 ms. Both delays will not be perceptible by users in practice.

The overhead of the controller of blanking audio when in exclusive mode affects audio recording applications. We ran an audio recorder software and measured the time spent in the controller. On average only 0.85% of the time is spent in the controller, showing that the overhead of the controller is indeed minimal.

7 Discussion

7.1 Improvements to the attack

We believe that sensory malware can take the following measures to improve its performance and stealthiness.

Stealthiness. To further reduce detectability, Soundcomber can choose the right time to analyze audio recordings:

[17] http://powertutor.org/

[18] http://developer.android.com/intl/fr/guide/developing/eclipse-adt.html

Table 3. Performance of speech and tone recognition

	Data length (seconds)		CPU time only (seconds)		Total time (seconds)		Power (mW)	Memory (MB)
	Mean	Std	Mean	Std	Mean	Std		
Speech	19.172	4.485	6.749	2.243	7.168	2.463	93.68	2.945
Tone	45.300	5.814	5.524	0.678	7.694	0.943	101.4	2.883

- *Analysis at night:* Soundcomber can defer processing a recording to an opportune moment when heavy CPU usage is less noticed, for example during the night or during longer periods of inactivity.
- *Analysis when user is not present:* the Android platform allows its applications to sense whether the user is present or not. Tracking the usage allows Soundcomber to find the right time to analyze recordings, so as to minimize the chance of being detected.
- *Analysis when charging:* prior research proposes to detect malware by tracking the power usage of a phone [8]. To evade this type of detection, Soundcomber can work on recordings only when the phone is charging and no power consumption information is available.
- *Throttle processing:* Soundcomber could easily be made to refrain from using up all CPU cycles, and instead only steal a small fraction of CPU time to analyze the recording. This makes detection of its presence even harder.

Performance. A more technical optimization is using fixed-point algorithms for DTMF detection. Current smartphones typically use ARM (Advanced RISC Machine) processors, which often do not include floating-point units, making floating-point operations expensive as they have to be emulated. Using fixed-point algorithms should increase DTMF detection significantly. We leave this exploration to future work.

Hotline detection. The algorithm used to detect the service hotline called can be improved. Brief experiments with another algorithm [14] (used by Shazam[19]) to determine which hotline was called seemed to be more accurate and efficient. We will study better detection techniques in follow-up research.

7.2 Defenses

We demonstrate the serious risk that sensory Trojan malware with even limited privileges poses to users. Sensitive information from a person's phone calls can be extracted stealthily and all known mechanisms are inadequate to stop the attack. In addition to the defense architecture we built, here we list some other measures that could be taken to mitigate this threat:

Tone playback settings. A simple defense against our tone-based attack is to to mute local playback of tones (available on some phones). While not normally selected by users, selecting it would prevent the tone-based attack, but not the speech-based attack.

Isolation. A simple defense would be to isolate the phone application by disallowing simultaneous access to any resource used by the phone application from other applications in the background. Unfortunately, this would also preclude legitimate applications such as call voice memo, speech translator and others.

Finer-grained sensor access. Given the sensitive nature of sensors such as the microphone and video camera, a more fine-grained permission model should be considered. For example, recording the microphone during a call could be a separate permission. Time in general could be used to regulate accesses (e.g., no recording during periods marked as meetings in the calendar) or potentially even the place (e.g., no recording at home).

Mediation of event management. Another problematic area highlighted by our work is the use of the event system as a covert channel. Android already has access control in place for some events, but this should be revisited and tightened, by monitoring and restricting (or having finer-grained control over) event flows between applications.

Anomaly detection. Anomaly detection could identify unauthorized uses of the microphone. Unfortunately, some applications might legitimately need access to the microphone even during phone calls, such as an application which records all phone calls for archival.

Network monitoring. Monitoring the network for anomalous traffic is unlikely to identify the Deliverer app because of the small amount of information that is sent over the network. This information can be included as an unnoticeable overhead in addition to its normal communication with the remote server (recall that the Deliverer app has legitimate Internet access). Obfuscated communications (e.g., through encryption) eliminate the possibility of detecting credit card numbers being leaked over the network interface.

[19] http://www.shazam.com/

8 Related Work

Using sensors such as the microphone [1] and video [15] to capture secret information has been studied in prior research. For example, Xu et al. [15] present a data collection technique using video cameras embedded in 3G smartphones. Their malware (also installed as a Trojan) stealthily records video and transmits it using either email or MMS. However, it does not automatically extract relevant information from the video recording and only limited processing of the captured data is done on the phone, information extraction is offloaded to a colluding server, which is not stealthy. In contrast, Soundcomber performs an efficient data analysis locally and transmits much less information (tens of bytes as opposed to video files). Also, with Soundcomber the malware master is not bombarded with numerous videos or data files from infected smartphones that need processing. Soundcomber distributes the computation onto those phones itself, and the malware master receives *only a small amount of high-value information*. As another example, Cai et al. [4] highlight the problems of more and more capable smartphones with sensors such as microphone, camera and GPS and how they can be used to snoop on the user. They also propose (but did not implement) a framework that could mitigate such threats, which involves black-/whitelisting and information flow tracking using taint analysis. Such a defense framework could have only limited effect on a malware like Soundcomber: for example, tracking taint propagation through covert channels is known to be difficult.

There are several approaches for detecting malicious applications on smartphones [3, 8]. For example, Bose et al. [3] propose behavioral detection of malware by monitoring system events and low-level API calls of an application on the Symbian platform. To classify applications, Support Vector Machines (SVM) are used and trained with both malicious and normal behavior of applications. Behavioral detection is promising in general; it is less clear, however, how a Trojan application would be classified, if the overt functionality mimics the behaviors deemed legitimate under SVM.

Another approach by Liu et al. [8] relies on monitoring the power consumption of a smartphone. This is only practical, though, if the smartphone is running on battery; Soundcomber can defer work to when the phone is charging. It also relies on hardware power-consumption monitors (not present on Android). Kim et al. [7] present a similar approach.

Instead of detecting malware after it is already present on the device, Enck et al. [5] propose to analyze the permissions requested by an application (specifically for Android) when the application is first installed. Rules, which specify what combinations of permissions are admissible, allow or prevent the installation. In our work we managed to separate the necessary permissions over two applications and let them communicate covertly, evading the security policy enforced by this approach — each application uses a "reasonable" set of permissions that in conjunction are dangerous. We also demonstrate that malware can use a legitimate application to deliver data it stole. Follow-up research by Ongtang et al. [10] not only examines permissions during the installation but also monitors their use during runtime, based on location and time, for example. Semantically rich policies define permissible interactions between applications. This, however, does not block covert channels, which are used by Soundcomber.

Other research has focused on the effect of cellular botnets on the network core, such as the research by Traynor et al. [13]. Our work, on the other hand, focuses more on how malware can extract information about individuals.

9 Conclusion

In this paper, we report our research on sensory malware, a new strain of smartphone malware that uses on-board sensors to collect private user information. We present Soundcomber, a stealthy Trojan with innocuous permissions that can sense the context of its audible surroundings to target and extract a very small amount of high-value data.

As sensor-rich smartphones become more ubiquitous, sensory malware has the potential to breach the privacy of individuals at mass scales. While naive approaches may upload raw sensor data to the malware master, we show that sensory malware can be stealthy and put minimal load on the malware master's resources. While we provide a defense for Soundcomber, more research is needed to control access to other types of sensor data depending on the context in which such data is being requested. We hope that our work with Soundcomber has highlighted the threat of stealthy sensory malware to stimulate further research on this topic.

Acknowledgments

We thank Zhou Li for his helpful comments. This research was funded in part by the National Science Foundation under grants CNS-0716292 and CNS-1017782.

References

[1] D. Asonov and R. Agrawal. Keyboard acoustic emanations. *Security and Privacy, IEEE Symposium on*, 0:3, 2004.

[2] Bell and LaPadula. Secure Computer Systems: Unified Exposition and Multics Interpretation. Technical Report ESD-TR-75-306, MTR 2997 Rev. 1, The MITRE Corporation, Mar. 1976.

[3] A. Bose, X. Hu, K. G. Shin, and T. Park. Behavioral detection of malware on mobile handsets. In *MobiSys '08: Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 225–238, New York, NY, USA, 2008. ACM.

[4] L. Cai, S. Machiraju, and H. Chen. Defending against sensor-sniffing attacks on mobile phones. In *MobiHeld '09: Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, pages 31–36, New York, NY, USA, 2009. ACM.

[5] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, pages 235–245, New York, NY, USA, 2009. ACM.

[6] ITU-T. Recommendation Q.23, 1994.

[7] H. Kim, J. Smith, and K. Shin. Detecting energy-greedy anomalies and mobile malware variants. In *Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 239–252. ACM, 2008.

[8] L. Liu, G. Yan, X. Zhang, and S. Chen. Virusmeter: Preventing your cellphone from spies. In E. Kirda, S. Jha, and D. Balzarotti, editors, *RAID*, volume 5758 of *Lecture Notes in Computer Science*, pages 244–264. Springer, 2009.

[9] J. Millen. Finite-state noiseless covert channels. In *Proceedings of the computer security foundations workshop II*, pages 81–86, 1989.

[10] M. Ongtang, S. E. McLaughlin, W. Enck, and P. D. McDaniel. Semantically rich application-centric security in android. In *ACSAC*, pages 340–349. IEEE Computer Society, 2009.

[11] A. V. Oppenheim and R. W. Schafer. *Digital Signal Processing*. Prentice–Hall, 1975.

[12] J. Proakis and D. Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications*, pages 480–481. Upper Saddle River, NJ: Prentice Hall, 1996.

[13] P. Traynor, M. Lin, M. Ongtang, V. Rao, T. Jaeger, P. McDaniel, and T. La Porta. On cellular botnets: Measuring the impact of malicious devices on a cellular network core. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 223–234. ACM, 2009.

[14] A. Wang. The shazam music recognition service. *Commun. ACM*, 49(8):44–48, 2006.

[15] 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 *WiSec '09: Proceedings of the second ACM conference on Wireless network security*, pages 69–78, New York, NY, USA, 2009. ACM.