Android Privacy C(R)ache: Reading your External Storage and Sensors for Fun and Profit

Stylianos Gisdakis, Thanassis Giannetsos, Panos Papadimitratos
Networked Systems Security Group
KTH Royal Institute of Technology
Stockholm, Sweden
{gisdakis, athgia, papadim@kth.se}
{*gisdakis*, *athgia*, *papadim*}@kth.se

ABSTRACT

Android's permission system empowers informed privacy decisions when installing third-party applications. However, examining the access permissions is not enough to assess privacy exposure; even seemingly harmless applications can severely expose user data. This is what we demonstrate here: an application with the common READ_EXTERNAL_STORAGE and the INTERNET permissions can be the basis of extracting and inferring a wealth of private information. What has been overlooked is that such a "curious" application can prey on data stored in the Android's commonly accessible *external storage* or on unprotected phone sensors. By accessing and stealthily extracting data thought to be unworthy of protection, we manage to access highly sensitive information: user identifiers and habits. Leveraging data-mining techniques, we explore a set of popular applications, establishing that there is a clear privacy danger for numerous users installing innocent-looking and but, possibly, "curious" applications.

1. INTRODUCTION AND BACKGROUND

The Android permission system was a significant step towards addressing the privacy concerns about the sensitive user information collected from third-party applications. It requires users of non-rooted devices to scrutinize permission requests made by applications. This way, they make informed decisions in an all-or-nothing approach [1]; they either accept all permissions or they opt out and do not use the application [2]. Such decisions, in turn, have an impact on their privacy since granting the requested permissions reveals different pieces of their personal information. Nonetheless, the meaningfulness of the provided feedback is controversial; studies [3] have demonstrated that the majority of Android users do not pay attention to, or understand, such permission prompts especially when they prevent immediate gratification (e.g., when users have decided to install an application) [4].

Furthermore, although Android permissions allow a (coarse-grained) regulation of an application's access to private information, they fail to provide comprehensive insights on how this information is actually used [5]. Even worse, in many cases, the access rights requested by applications are inordinate to the functionality they offer [6]. To mitigate the consequences of permission over-provisioning, state-of-the-art privacy research proposes mechanisms that utilize static analysis [7], or retrofit the Android runtime environment [5] with privacy controls over the data applications access. Other works suggest substituting private and sensitive information with innocuous shadow data [8], or deterring the transmission of sensitive information over the network [9].

At the same time, there has been awareness in the community that external storage can have useful/sensitive information [10], it is still not considered a significant concern [11]. This work shows that it is a mistake to consider data in *external memory* unworthy of protection. All it takes is to lure unsuspecting users to install seemingly innocent but "curious" applications that simply accesses the external storage.

On top of that, these data may be augmented with smartphone sensor data; mobile devices are increasingly equipped with a range of sensors such as GPS, microphone, accelerometer, light, and proximity sensors. These sensors can be effectively used to infer a user's context including his location, transportation mode, social state, etc [12]. Although some of these hardware components (e.g., microphone, camera) are mediated and protected by the operating system (through permission requests), others require no permission and are accessible by any third-party application via intents and deputy apps [13]. In this work, we show that even with a simple analysis of any values retrieved from such "*unprotected*" sensors, one can get valuable insights about a user activities.

Contributions: We consider applications for which Android's play market states that *"this application does not require any special permissions"* and we show how they can act as gateways into users' personal data. From a corpus of the most downloaded (free) Android applications (e.g., social networking, messaging, infotainment, navigation, etc.), we demonstrate how someone can: (*i*) extract private user information (user identifiers, activities, habits and location) and (*ii*) create an accurate profile of applications and activities, executed on the host device, simply by examining what these popular applications store on the commonly accessible external storage and by leveraging the phone's unprotected sensors. We employ data-mining techniques ranging from face recognition to natural language processing, (*iii*) to attain other highly sensitive information (list of contacts and phone numbers), without requesting the appropriate "dangerous" permissions (as these could arouse suspicion). *The contribution of this work is to highlight that user privacy does not only depend on the understanding of the granted permissions but also on the cautious usage of the commonly accessible and unprotected sources of information.* Indeed, forensics analysts, with physical access to the device, have (authorized) access to any type of stored information. Nonetheless, in this work we show that even the simplest of zero-permission applications can do the same. *To the best of our knowledge, this is the first work to concretize and investigate the threat the community suspected and yet has not deemed a pressing matter.*

Focus: Table 1 enumerates all the Android applications, considered in this work, along with the number of user downloads and their respective market scores; these values are indicative of their *popularity* and *trust* as perceived by the

Application	Downloads	Market Score
Facebook	1000 - 5000 (M)	4.0
WhatsApp	500 - 1000 (M)	4.4
Skype	500 - 1000 (M)	4.1
Twitter	100 - 500 (M)	4.1
Instagram	100 - 500 (M)	4.5
Viber	100 - 500 (M)	4.3
Snapchat	100 - 500 (M)	4.0
WeChat	100 - 500 (M)	4.3
Tango	100 - 500 (M)	4.2
LinkedIn	10 - 50 (M)	4.2
Vine	10 - 50 (M)	4.2
Foursquare	10 - 50 (M)	4.0
Swarm	5 - 10 (M)	3.8
Spotify	50 - 100 (M)	4.5
YouTube	1000 - 5000 (M)	4.1
Flipboard	100 - 500 (M)	4.4
PushBullet	1 - 10 (M)	4.4
Dropbox	100 - 500 (M)	4.5
AirDroid	10 - 50 (M)	4.5
NavFree	10 - 50 (M)	4.1
Be-On-Road	10 - 50 (M)	4.1
GPSTest	5 - 10 (M)	4.4
OneDrive	10 - 50 (M)	4.4

Table 1: Free Android applications considered in this work

users [14]. Our focus is on the kind of data that may be the target of severe privacy violation, notably: (*i*) Communication data, such as SMS, MMS, Voice messages, etc., (*ii*) Media data, from the camera or microphone (*iii*) Location data, i.e., GPS data (exact location) or location inferred from the users' social network and geotagged photos, (*iv*) Text data, including documents and emails among others, (*v*) Contacts, i.e., the users' list of contacts extracted either by the smart-phone's address book or by the social media that the user participates in, (*vi*) History/Usage data indicating the user's preferences, (*vii*) Identity data; i.e., all the unique identifiers that can be leveraged to identify a user, e.g., device ID, email address, profile ID, etc., and (*viii*) Contextual data that can be used to infer a user's context and activity (i.e., by leveraging unprotected phone sensors such as accelerometer, gyroscope, magnetometer and temperature sensors).

Nonetheless, we do not solely focus on the types of sensitive information that can be directly extracted from a smart-phone but also on the analysis mechanisms that can be applied in order to ex-filtrate as much information as possible.

The rest of this paper is organized as follows: Sec. 2 and 3 present the analysis framework and the inference process we perform over the data acquired from the smart-phone. Sec. 4 presents our findings alongside with an evaluation of the employed inference mechanisms. Sec. 5 discusses possible mitigation mechanisms. Finally, Sec. 6 concludes this work.

2. FRAMEWORK DESIGN

Our data extraction and analysis framework comprises two parts: an Android client running on the user's smart-phone and an analysis server that performs the off-line analysis and inferences over the ex-filtrated user data.

Android Client: Our client requests the two most common, and seemingly harmless, permissions [15]: the INTERNET and READ_EXTERNAL_STORAGE permissions which are given by default to applications running on newer Android versions [16]. Simply put, when installing such a mobile application, users will be prompted with a (comforting) message stating that *"this application does not require any special permissions"*. The client traverses all files, residing on the external storage, and sends them to the server for further analysis; it does not perform any local data analysis on the device. It also accesses the unprotected phone sensors and sends the retrieved values to the server for processing (Sec. 3.3).

To enhance the *stealthiness* of this client-server communication, one could program the application to transmit data only when the phone is connected to some Wi-Fi network (i.e., by requesting the ACCESS_WIFI_STATE permission). The rationale is to prevent extensive usage of the cellular bandwidth because this could serve as an indication of disproportional, for the purpose of the application, network usage. Note, however, that this permission is not necessary for our design because it is considered intrusive. Moreover, the client can encrypt the data it transmits to the server in order to avoid possible traffic analysis from any existing information-flow tracking system [5]. It can also operate during periods that the phone is "*inactive*". To do this, we leverage the accelerometer sensor to examine whether the device remains motionless for long periods of time (e.g., when the phone is placed horizontally during early morning hours).

The aforementioned functionality is *covert*; if such an application would be deployed on the market it would also have to offer a service to users so that they use it. Nevertheless, the *overt* functionality is beyond the scope of this work.

Analysis Server: After receiving the various data files (e.g., text, image, audio), the analysis server performs the off-line analysis for extracting sensitive information.

3. FILE ANALYSIS

3.1 Image Files

Images constitute one of the most common type of files stored on Android's external storage. They range from application icons and photos exchanged by users (through social networking and instant messaging applications) to photos taken by the smart-phone's cameras (i.e., located in the DCIM folder). Depending on the application that generated an image (this can be easily inferred by the path of the file), we analyze the files for the following types of information.

3.1.1 Face Detection

Images retrieved from the DCIM folder are (most probably) photos taken by the user. Therefore, we initially run a face recognition algorithm to detect (possible) faces. To do this, we employ the implementations of well-known algorithms from the OpenCV[1] platform. We used pre-trained Local Binary Pattern (LBP) classifiers included with the OpenCV library for face detection. In case we confirm the existence of faces, we extract the EXIF (Exchangeable Image file Format) meta-data from the image file. These contain copious information of interest: timestamps of when the photo was captured and camera identifiers that reveal which phone camera was used.

Leveraging these meta-data, we first focus on the photos taken with the front-facing camera of the device (i.e., the value 0 of the $FNumber$ attribute reveals this). Since the most common use of this type of camera is for self-portrait pictures, our goal is to identify any photos that capture the user's face. On the other hand, for photos taken using the back facing camera, we examine the phone rotation at the time the photo was captured. More specifically, we look at the *Image Direction* field of the EXIF meta-data, which contains the camera's rotation when the photo was taken. This is calculated using the smart-phone's embedded sensors (i.e., accelerometer, gyroscope and magnetometer) and it can give

[1]http://opencv.org

us a good insight on the content of the photo: if the rotation value is within the [160, 190] degrees interval then we can (with high probability) assume that the photo contains the user's face (e.g., photos commonly known as "*selfies*"). In both cases, these photos are of particular interest because they may contain user identifying information.

Besides simple face detection, the analysis of such face-containing images also enables the extraction of, ample, user profiling information. By leveraging face characterization algorithms, we can further infer the user's age (i.e., a coarse-grained age), genre, race, face attributes (e.g., beard and mustache) and emotional status (when the photo was taken). To do this, we employ Face++ API[2] which provides facial analysis algorithms (see Sec. 4).

3.1.2 Location Extraction

Android provides an option for geotagging photos, i.e., saving the geographical coordinates of the device when the photo was taken. These coordinates are stored inside the GPS "*Latitude*" and "*Longitude*" fields of the EXIF metadata. If no such location information is available, we try to make a best-effort approximation by applying a *reverse Google Image search*. The result of such a search contains various pieces of information describing the queried image, including a "best guess" field; a text description of the image in question. In most cases, the best guess is the name of the monument/area/landmark captured by the picture (Sec.4.2).

3.1.3 Social Interests Inference

To ex-filtrate a user's social interests, we leverage images created by social networking applications and newsreaders. As an example, consider Twitter whose mobile client caches (inside a dedicated directory in the external storage) the avatars of all people a user "*follows*". The usual purpose of these avatars is to convey short information about their owners. Therefore, we can leverage such images in order to infer some of the user interests and likings.

First, we obtain a text description of the image from the best guess result of Google's reverse image search. Based on the outcome, we then retrieve keywords associated with the given text by querying "DBpedia.org".[3] To exemplify this, assume a user follows the official account of the CNN news agency. The avatar used by the CNN account (stored on the SD card) is the CNN logo. Employing the reverse image search gives us the term "cnn app" which, in turn, when queried in DBpedia yields a number of associated terms such as: television channel, basic cable and news channel.

Similar to Twitter, there are a number of other social networking applications that can be used to ex-filtrate information about a user's online social networks. Amongst the most popular ones is LinkedIn, a business-oriented social networking service. LinkedIn users create business profiles and establish "connections". LinkedIn's application stores (on the SD card) the profile pictures (or logos) of the people and companies a user follows. By running the previously described face detection algorithm (Sec. 3.1.1), we can identify and sift profile photos from other non-interesting images (e.g., corporate logos). For the former, we then run a reverse image search and retrieve the first 10 (for example) returned results which (in most of the cases) point to the name of the user. The same holds for location-based services such as Foursquare and, thus, the same approach can be applied. Combining all such social-related information enables us to construct parts of the user's social network.

We use LinkedIn profile images because, given the purpose of this service, it is likely that users upload high quality professional profile photos. This facilitates the face detection and recognition processes to produce better results.

3.2 Audio and Voice Messages

Most messaging applications supporting audio communications (i.e., voice messages) store unencrypted copies of all audio messages a user sends or receives. This is surprising considering the amount of privacy violating information that may be extracted from such data. Nonetheless, leveraging audio data to extract useful information is far from trivial. Towards that, our framework provides the following steps:

Speech-To-Text (STT) translation: First, we convert the audio file from the Advanced Audio Coding (AAC) format, which is used by most messaging applications, to the Waveform Audio File (WAV) format. This is done through the libav[4] open source library, which provides cross-platform tools to convert, manipulate and stream a wide range of multimedia protocols. Such a transformation allows the subsequent translation of the audio file to text by leveraging AT&T's STT library [5].

Part-of-Speech (PoS) tagging: The generated text transcript is given to a PoS tagger. This performs a syntactic analysis, identifying the lexical categories (e.g., nouns, verbs, adjectives) of the text produced during the STT phase.

These two pre-processing steps set the baseline for a deeper inspection of the provided audio files. We focus on: (*i*) location, (*ii*) time and (*iii*) user-referring information.

3.2.1 Location, Time & Named Entities

To extract possible location information from audio data, we examine the produced text transcript for *prepositions of space*; i.e., "in", "on", "to" and "at". Once identified, we then look for any singular or plural nouns (tagged as "NN" after the PoS tagging phase) contained in the text. As an example, consider the text phrase "Let's meet at Sunset Boulevard" retrieved from a WhatsApp voice message. The existence of the "at" preposition is indicative of location.

To improve the performance of our analysis, we leverage the various types of verbs that may be used in a phrase. Active verbs denoting motion serve as evidence of location-specific information (for instance "*Driving to* Manhattan"). Moreover, passive verbs describing the user's *activity context* (i.e., waiting, to be, meet) can be effectively used to infer her location. To automate the location extraction process, we employ Stanford's Named Entity Recognizer[6] (NER), a library that provides text processing capabilities for extracting various sentence features (e.g., in our case location). Location can refer to cities, states, countries, etc.

The analysis of the PoS tagged transcript also provides time-related information of user activities. We search for time prepositions: "at", "in" and "on". Their identification is now based on the contained time indexes (or intervals) of described events; e.g., "Let's meet at 1 pm". Once more, we employ Stanford's NER to extract time-specific information.

[2]http://www.faceplusplus.com/

[3]DBpedia (`http://dbpedia.org`) is a community effort to extract structured information from Wikipedia.

[4]http://libav.org/

[5]`http://developer.att.com/sdks-plugins/speech-for-android`

[6]http://nlp.stanford.edu/software/CRF-NER.shtml

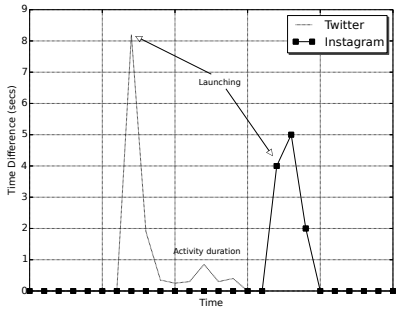


Figure 1: Time-stamp Analysis.

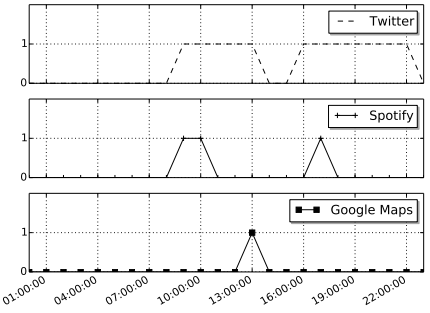


Figure 2: Profiling Application Activity.

We also examine audio files for named entities [17]. Stanford's NER comes with "named entity" recognizers for the English language. In our work, the word "named" restricts the task to only those entities for which one or many designators have been defined. For instance, "I am with Mary! Will you join?" is referred to as "Mary" whereas "I work at Google!" is referred to as "Google". Rigid designators include proper *person* names (i.e., Nick, Mary, etc.) as well as certain popular *organizations* (i.e., Google, Skype, etc.).

3.3 Inferring User Context

As described in Sec. 2, our framework comprises an inference engine that enables adversaries to target the user's contextual privacy; issue queries for extracting the user's context (e.g., sleeping, walking, training, etc.). Towards that, the mobile client can access the phone sensors and send any retrieved values to the analysis server. Recall that only the phone camera and microphone require specific access permissions. The rest of the sensors are "unprotected" meaning that accessing them is not noticeable by the user. This can be done either directly or indirectly via intents and deputy apps [13]; Android intents allow applications to access resources indirectly by using deputy apps that have access to the requested resource or information [18]. This is possible because intents can be used for both inter- and intra-application invocations.

To infer user context we leverage *random forests* [19]: collections of decision trees, each trained over a different bootstrap sample. We obtained the training set (i.e., sensor values labeled with activities) from the PAMAP[7] dataset which contains sensor readings (i.e., accelerometer, gyroscope, magnetometer) from 17 subjects performing 14 different activities (e.g., walking, cycling, laying, ironing, computer work). We considered only a subset of the including sensor types focusing on those that are already available in current smart-phones: temperature (Samsung Galaxy S4 has a dedicated temperature sensor), accelerometer, gyroscope and magnetometer.

Each decision tree of the random forest is a classification model created during the exploration of the training set. The interior nodes of the tree correspond to possible values of the input data. For instance, an interior node could describe the values a sensor s_1 gets (e.g, $s_1 > \alpha$). Nodes can have other nodes as children, thus, creating decision paths (e.g., $s_1 > \alpha$ *and* $s_2 < \beta$). Tree leaves mark decisions (i.e., the classifications) of all training data described by the path from the root to the leaf. For example, samples for which sensors $s1$ and $s2$ take the values $s_1 > \alpha$ *and* $s_2 < \beta$ refer to the *walking* activity. After the training phase is completed, the classifier infers the user's context based on the sensor values sent by the mobile client. More specifically, the client monitors the phone's sensors for a predefined period of time and submits them to the analysis server. Upon reception, these values are examined by the classifying ensemble and a decision on the possible user context is produced.

3.4 Profiling Application Usage

Before Android Lollipop, applications granted with the GET_TASKS permission could query the ActivityManager and retrieve all running tasks via the *getRunningTasks()* method. For Android Lollipop, Google decided to deprecate this functionality for *privacy* reasons as *this method could leak personal information to the caller* [20]. Furthermore, according to the Android team [21], this method has been a significant attack vector for malicious apps allowing them to examine the types of applications users employ or to launch malicious actions when users perform specific actions.

Despite this newly introduced restriction, a simple examination of the *last-modified time-stamp* of specific files and folders renders feasible the profiling of currently running applications. More specifically, as it has been discussed so far, different applications access different files residing on the phone's external storage. Any malicious application monitoring specific files can detect *if, when* and for *how long* an application is used. For example[8], by monitoring the last-modified time-stamp of the file *com.twitter.android/cache/users/journal*, a malicious application can detect if Twitter is running. Similarly, the file *com.instagram.android/cache/video/journal* gives information on whether the user has launched Instagram. Our mobile client frequently retrieves (e.g., every 3 *sec*) the last-modified time-stamp of the application-specific file. For example, let ts_i and ts_{i-1} be the time-stamps of an application specific file retrieved at times i and $i-1$. If $dt = ts_i - ts_{i-1} > 0$ then this serves as an indication the the application is active and is accessing the application specific file. Fig. 1 plots dt as a function of time. As the figure shows, it is easy to detect when the application is launched since when this happens the two consequent last-modified time-stamps differ. Moreover, examining the time-interval for which $dt > 0$ is an indication of the usage duration.

We leverage time-stamps to accurately profile a user throughout long periods of time. Fig. 2 illustrates the usage patterns of three applications retrieved from the phone of one of the authors for a period of 24 hours. We plot time on the $x-axis$ and a boolean value on $y-axis$ indicating whether the application has has been activated at a given hour (done by retrieving

[7] `http://www.pamap.org/demo.html`

[8] Due to space limitations we provide some examples of application specific files monitored by our system.

the last-accessed time-stamp). Of course, the mobile client can monitor application specific files at any granularity (e.g., every 15 *mins*). For Spotify the client monitors the file */storage/sdcard0/Android/data/com.spotify.music/files/spotifycache/mercury.db* and for Google Maps the file */storage/sdcard0/Android/data/com.google.android.apps.maps/cache/cache_its.m* which is updated whenever the application is invoked.

4. ANDROID APPLICATION ANALYSIS

This section presents our findings from the extraction of user private information from the phone's external memory. Towards that, we examine closely the data that are generated by the chosen Android applications (Table 1) available in the market. We then proceed with the assessment of our framework's accuracy and inference mechanisms.

Instant Messaging Applications: We begin our analysis by looking *Viber*. Viber keeps a configuration file with information on the user identity, profile picture and phone number. In particular, inside the Viber directory, the hidden file *.userdata* contains the username specified when registering with the messaging service. This identifier is often the user's real name. Furthermore, the *canonicalized_number* field (in the same file) carries the user's phone number along with the international call prefix for the specific country (based on the mobile operator the device is registered with).

Most messaging applications encrypt text-based communications before storing them on the external storage. The only exception is the *Tango IM*, which stores clear-text messages and conversations in the *com.sgiggle.production* folder. Exchanged media (e.g images and audio files) are, in many cases, stored unprotected. Viber caches voice messages under the folder *.ptt* located in the application's main directory (i.e., *SDCard/Viber*). Similarly, *WeChat*[9] leaves all audio, video and image files unprotected. In addition, WeChat enables users to share their location by sending a map in the form of an image. These images are also stored on the SD card, thus, enabling the extraction of the user's location.

Besides storing photos, audio messages and videos unprotected, *Skype* provides (by far) the most privacy sensitive information to any curious entity. Whenever users employ the "automatically add friends" functionality, Skype's Android client generates an SQLite database which is essentially an exact copy of the user's address book. For instance, for a user with skype_id "Bob" the list of contacts is stored inside the file "*com.skype.raider/bob_ingestion.db*". As one could understand, this has severe privacy repercussions: an application with (simple) read access privileges to the external storage can essentially retrieve all the phone numbers and email addresses in the user's contact list.

Music Streaming Applications: *Spotify* keeps all configuration files inside the *com.spotify.music* folder. The internal <user-id>-user sub-folder holds the user's Spotify ID (in its path name). We can leverage this identifier to retrieve information about the user, by issuing a GET request to *https://api.spotify.com/v1/users/<user-id>*. This returns a JSON object with ample user-specific information, such as the user display name, the number of followers she has and her profile picture. Moreover, Spotify allows users to connect (and authenticate) using their *Facebook* accounts. In this case, the returned display name is the user's Facebook ID which, in most cases, is her name and surname (while the user's profile picture is her Facebook avatar). Examining whether a user has federated her Spotify and Facebook accounts can be done by checking the *facebook_connected* field contained in the *orbit.settings* file. This file can also be used to extract information regarding the user's *LastFM* account, another music streaming service that can be connected to Spotify.

[9]Popular IM application in Asian countries.

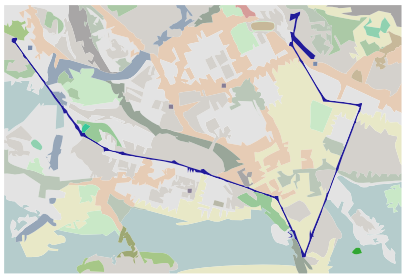


Figure 3: Traces extracted from Google location history.

The *local-files.bnk* file contains the names of all the songs a user has downloaded along with the respective paths of the audio files. Moreover, inside the *cosmos_player_state* file, we can find the user's latest song search. The file *offline_lists.bnk* contains all the playlists saved by a user. If a playlist was created by a user's friend, we also get her identifier.

Social Networking Applications: In Sec. 3.1.3 we discussed how the images stored by *Twitter's* Android client can be leveraged to infer information about the user. Additional information can be extracted by examining the data Twitter stores on the phone's external storage. More specifically, Twitter generates a binary file under the name "abd_<user_id>" (where <user_id> is the user's account id) inside the *com.twitter.android* folder. Knowing the user's account id allows us to issue the following request to Twitter's API: *https://api.twitter.com/1.1/users/show.json?user_id =<user_id>*. Assuming that the user does not have a protected account, this request will return the number of followers a user has, her friends, the number of status updates, her preferred language and her *retweets* with exact time-stamps and links (i.e., mentions) to other user profiles.

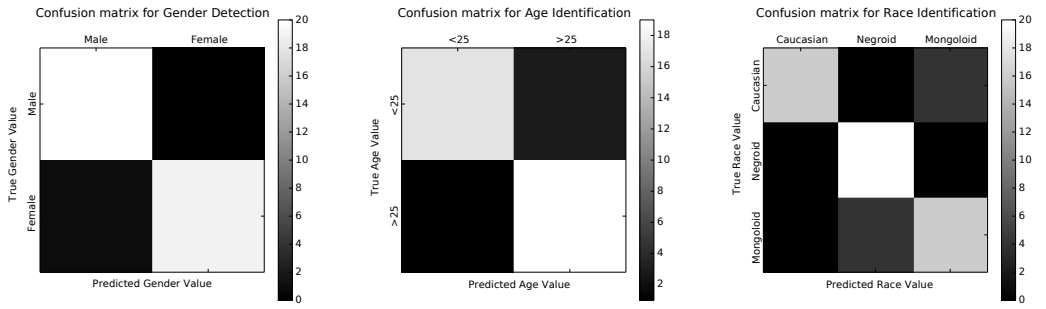
Vine stores inside the *co.vine.android* folder files (the exact file names differ for each user) containing the user's username, her id, her phone number, her email address and information about connected accounts (e.g., Facebook, Twitter). Another file, in the same folder, stores the information (email address and user ids) of the users she follows. Instagram stores all the video files a user browses on the phone's external memory.

Navigation Applications: The *NavFree* crowd-sourced navigation application stores application-specific data inside the *com.navfree.android.OSM.ALL* folder. The internal subfolder *files/UserData* has the *Favorites.txt* file that stores all the points-of-interest (POIs) a user has saved, along with their geographical coordinates. In the same folder, one can also find the *Home.txt* file, which contains the geographical coordinates of the user specified home location.

Be-On-Road, another navigation service, stores inside the "Favorites" folder (residing on the *cz.aponia.bor3* path) the geographical coordinates of all the POIs, including home and work locations and all the addresses a user has searched for.

Finally, the *GPS Test* navigation application, which provides a utility for showing the signal strength of in-range satellites, saves a log file on the external storage that contains all user's visited locations along with time-stamps.

Cloud Storage Applications: *Microsoft OneDrive* caches



(a) (b) (c)

Figure 4: Gender (a), age (b) and race (c) detection from user photos.

on the external memory the files a user downloads (inside the *com.microsoft.skydrive* directory). The internal sub-folder *instrumentationApplicationInsightsV1Channel* keeps detailed log user uploads. These files contain additional information such as time-stamps, device model and the Android version.

Similarly, *Dropbox* saves on the external memory all files a user uploads or downloads. Moreover, detecting the Dropbox account id can be done by examining the name of the folder inside the path *com.dropbox.android*.

AirDroid is a popular application allowing users to manage their Android phones and tablets via their PCs. Inside the directory "*com.sand.raider/com.sand.airdroid/files*" the file main.log contains the following pieces of information: (*i*) the phone's *IMSI*, (*ii*) the phone's IP address, (*iii*) the network connection type (i.e., wireless or cellular), and (*iv*) the user's email address and her name. In case the user employs Google as the authentication provider, then the extracted name field is the user's actual name (i.e., first and surname). Moreover, *AirDroid* also allows users to locate their phones. Enabling this functionality, allows the extraction of their fine-grained geographic location along with the time-stamp of when the query was issued (stored in the same main.log file)

PushBullet is a file sharing and messaging application. Inside the folder *com.pushbullet.android* the application stores a log file. This log file contains the email address with which the user registers to the service. Moreover, the same file holds information about the email addresses of a user's contacts.

Invoking Browsers: Although popular browsers do not store any sensitive information on the phone's external storage, we can still exploit them to extract particular user features. More specifically, we focus on users that have logged-in (with their browsers) to their Google profiles. Launching an *intent* for *https://maps.google.com/locationhistory/b/0/kml?startTime=<s>&endTime=<e>* (where <s> and <e> define the starting and ending date in "epochs") retrieves the user's location history (for that period) and stores it on the external storage. Of course, this requires that the user has enabled the location service on her mobile phone. The retrieved XML file (accessible by any application) contains the latitude and longitude coordinates along with detailed time information (Fig. 3). With this data, we can (for example) infer the user's home and work location.

4.1 Face Recognition and Characterization

We begin with an assessment of the face detection analysis presented in Sec. 3.1.1. Our goal is to detect user facial attributes by examining retrieved self-portrait photos (i.e., "selfies" captured from their smart-phones). We focus on three attributes: (*i*) user gender, (*ii*) age and (*iii*) race: they are important for face recognition-based marketing [22].

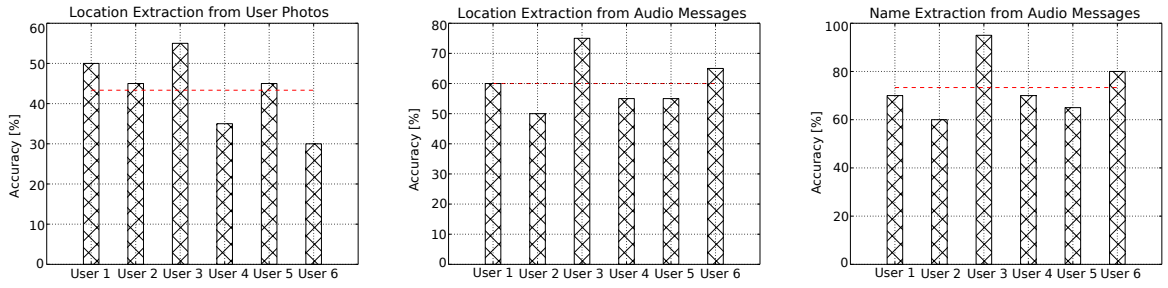
For our evaluation, we used photos from the SelfieCity[10] dataset, containing 3200 Instagram selfie photos of users with various lighting conditions, expressions and realistic backgrounds. Each photo is categorized according to different criteria (e.g., user age and gender). Furthermore, many of these photos contain filters (e.g., black-white), as usually is the case with the pictures users share on Instagram. We believe this dataset is similar to the photos one would expect to find stored on the external storage of average users' devices.

We present our findings, on correctly identifying a user's attributes, through *confusion matrices*. Each column of the matrix shows the "predicted" instances of a class (e.g., male or female), while each row shows the true class of the instances. The diagonal elements (in the matrices) show the number of correct classifications made for each class (i.e., true positives and negatives), and the off-diagonal elements indicate the errors (i.e., false positives and negatives).

Fig. 4 (a) depicts the gender classification results of 40 user photos (i.e., 20 male and 20 female users). As we can see, it is relatively easy to detect the user gender from user photos. In particular, the gender detection algorithm correctly identified all photos of female users and misclassified only one male. Next we move on to the age detection process (again by considering 40 user photos). In this case, we split users into two groups; one group comprises 20 photos of users within the $[17, 25]$ years old interval whereas the second group contains photos of users that are older than 25 years old. These age groups are considered to be important for the planning and execution of marketing campaigns [23]. As we can see in Fig. 4 (b), the Face++ algorithm is almost 100% successful in correctly identifying the user's age based on her photos. Indeed, 19 users below the age of 25 years old were correctly classified whereas only one user was falsely identified as older. Furthermore, 17 users in the above 25 years old class were classified correctly (i.e., 3 incorrect classifications). Finally, Fig. 4 (c) presents the race classification results for 60 individuals belonging to three different ethic groups (Caucasian, Negroid and Mongoloid). Once more, we observe that it is relatively easy to accurately identify a user's race.

Note, however, that the presented results do not intend to

[10] `http://selfiecity.net`



(a) (b) (c)

Figure 5: Location Extraction from user photos (a) and voice messages (b). Name extraction from audio messages (c).

evaluate the accuracy of the employed face detection software. Our goal is to highlight the ease of identifying user attributes (by analyzing photos stored on the external storage) with even readily available tools. All it takes is for the user to download a seemingly benign, but curious, application.

4.2 Analysis of Photos and Voice Messages

First, we assess the extraction of location information from user photos (Sec. 3.1.2). For this experiment, we make use of 120 photos taken by 6 users (i.e., 20 photos from each user) and we examine for how many of them we are able to extract fine-grained location-specific data. We sample photos taken with different types of mobile devices in order to demonstrate that our findings do not depend on the multiplicity or the kind of utilized hardware and software. These photos are capturing landmarks of different cities and countries.

As we can see in Fig. 5 (a), we are (on average) able to infer location information from 51 photos (i.e., 42.5% accuracy) by leveraging reverse Google image search. Note that users were requested to provide us with photos of their choice. As a result, some of them (e.g., user 1 and user 3) gave us pictures of famous landmarks of major U.S. or European cities whereas the rest provided photos of less famous landscapes. Moreover, we do not consider here the extraction of location information from the image meta-data as it is trivial.

Next, we evaluate the analysis of voice messages as presented in Sec 3.2.1. We refrain from assessing the accuracy of the STT translation because this is reflected in the performance of the employed natural language processing to detect location and named entities; better STT translation yields higher location and name extraction accuracy. First, we focus on extracting location information from the audio files. In particular, we asked six subjects (3 male and 3 female) to arrange a hypothetical meeting by sending 20 voice messages on WhatsApp. The meeting place was selected from a list of famous places in the U.S (e.g., Soho, Michigan Avenue, Central Park). The participating subjects come from different countries and, thus, have different English skills and accents. Fig. 5 (b) depicts our results: on average we extract correct location information from 60% of the voice messages. Although the output heavily depends on the English skills of each subject (e.g., subject 3 is a native English speaker), we manage to overall achieve high accuracy ($> 50\%$).

Finally, Fig. 5 (c) presents our findings on the extraction of named entities (from voice messages). As we can see, the achieved accuracy is significantly higher compared to the case of location extraction: names are, usually, single-word or short words and, thus, they can be detected easier.

4.3 User Context Detection

We evaluate the accuracy of the classifier ensemble focusing on the accuracy of user-context classification. For each evaluation, we select one subject of the PAMAP dataset, at random, as the classifier's training input. We then evaluate the ensemble's accuracy for the remaining subjects.

As depicted in Fig. 6 (a), the overall classification accuracy is above 50%. This serves as the best indication that we can, indeed, use various sensor readings to target user contextual privacy. Fig. 6 (b) illustrates the ensemble classification accuracy as a function of the sampling rate of the mobile client: as the sampling rate decreases, the classifier's accuracy drops. However, even for small sampling rates (1 sample every 45 *sec*), the classification accuracy still remains relatively high ($\simeq 50\%$). Finally, Fig. 6 (c) presents the informativeness of the employed sensor types with respect to user contexts. We express "sensor importance" as the (normalized) total reduction of uncertainty brought by that feature (i.e., the Gini index [19]). As it can be seen, magnetometers and gyroscopes are the most privacy intrusive sensors.

5. POSSIBLE MITIGATION STRATEGIES

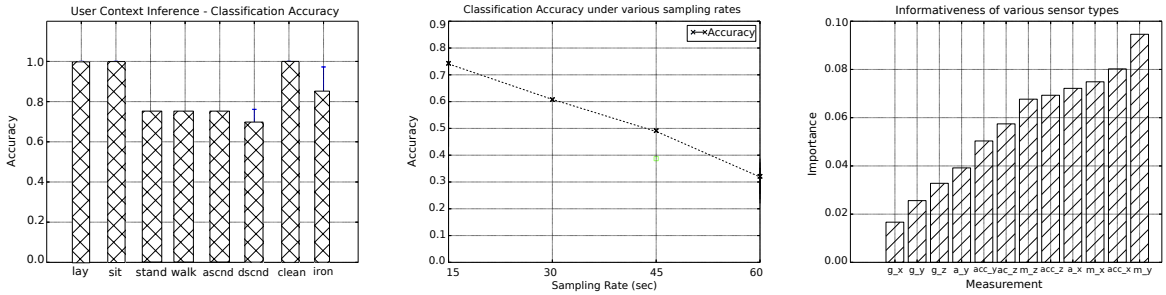
Google restricted in Android 4.4 *write access* to the external storage. More specifically, applications do not have write privileges to locations outside their owned area (i.e., /Android/data/application_name). This, essentially, prevents applications from manipulating data not belonging to them. Unfortunately, applying the same access control with respect to *read access* is far from trivial; it would affect, many applications (file managers, anti-viruses and backup software).

To reduce the privacy exposure of users is the proper handling of sensitive user information. More specifically, user photos can be protected by mechanisms such as the one described in [24], enabling user control over their photos; if users tag photos as sensitive, then they cannot be accessed by any other third-party application.

Moreover, audio and video communication files should be protected just as it is the case with text communications. Simply put, there is no point in encrypting text messages and leaving audio and video messages unprotected (e.g., as most messaging applications do). Of course, encryption of audio and video data should not significantly affect user experience. Similarly, configuration files should not be safely stored on the phone's internal memory.

6. CONCLUSIONS

This work demonstrated how simple and seemingly harmless Android applications can be used as gateways into our



(a) (b) (c)

Figure 6: Inferring User Context: (a,b) Classification Accuracy, (c) Sensor Evaluation.

personal data. We did this by focusing on what has been, so far, overlooked by the Android privacy community: the breadth and the wealth of the privacy-harming information stored on Android's external storage and the informativeness of unprotected phone sensors. Leveraging data-mining techniques and publicly available libraries we showed that user privacy depends on the cautious usage of commonly accessible memory locations. We hope this work will stimulate further research on this matter, especially because it is now obvious that (the true) user privacy exposure is up to any curious or malevolent actor or Android developer.

References

[1] Sanae Rosen et al. "AppProfiler: A Flexible Method of Exposing Privacy-related Behavior in Android Applications to End Users". In: *ACM Conference on Data and Application Security and Privacy*. San Antonio, Texas, USA, 2013.

[2] Eric Gustafson et al. "Quantifying the Effects of Removing Permissions from Android Applications". In: *IEEE Mobile Security Technologies (MoST)*. San Francisco, CA.

[3] Alexios Mylonas et al. "Assessing Privacy Risks in Android: A User-Centric Approach". In: *Risk Assessment and Risk-Driven Testing - First International Workshop, RISK*. 2013.

[4] Alessandro Acquisti and Jens Grossklags. "Privacy and Rationality in Individual Decision Making". In: *IEEE Security and Privacy* 3.1 (Jan. 2005), pp. 26–33.

[5] William Enck et al. "TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones". In: *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*. Vancouver, Canada, 2010.

[6] Tom Fox-Brewster. *"Check the permissions: Android flashlight apps criticised over privacy"*. Oct. 2014. URL: `http://www.theguardian.com/technology/2014/oct/03/android-flashlight-apps-permissions-privacy`.

[7] Manuel Egele et al. "PiOS : Detecting privacy leaks in iOS applications". In: *NDSS 2011, 18th Annual Network and Distributed System Security Symposium, 6-9 February 2011, San Diego, CA, USA*. San Diego, UNITED STATES, Feb. 2011. URL: `http://www.eurecom.fr/publication/3282`.

[8] Yajin Zhou et al. "Taming Information-stealing Smartphone Applications (on Android)". In: *Proceedings of the 4th International Conference on Trust and Trustworthy Computing*. Pittsburgh, PA, 2011.

[9] Peter Hornyack et al. "These Aren't the Droids You'Re Looking for: Retrofitting Android to Protect Data from Imperious Applications". In: *18th ACM Conference on Computer and Communications Security*. Chicago, Illinois, USA, 2011.

[10] Wired. *Break Out a Hammer: YouâĂŹll Never Believe the Data âĂŸWipedâĂŹ Smartphones Store*. URL: `http://www.wired.com/2013/04/smartphone-data-trail/`.

[11] Dave Smith. *"On the Edge of the Sandbox: External Storage Permissions"*. Mar. 2014. URL: `http://possiblemobile.com/2014/03/android-external-storage/`.

[12] Michaela Götz et al. "MaskIt: Privately Releasing User Context Streams for Personalized Mobile Applications". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Scottsdale, Arizona, USA, 2012.

[13] Xuetao Wei et al. "ProfileDroid: Multi-layer Profiling of Android Applications". In: *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*. Mobicom. Istanbul, Turkey, 2012.

[14] Patrick Gage Kelley et al. "A Conundrum of Permissions: Installing Applications on an Android Smartphone". In: *Proceedings of the 16th International Conference on Financial Cryptography and Data Security*. Bonaire, 2012.

[15] Adrienne Porter Felt et al. "Android Permissions: User Attention, Comprehension, and Behavior". In: *Proceedings of the Eighth Symposium on Usable Privacy and Security*. Washington, D.C., 2012.

[16] Charles Arthur. *"Boot up: more Android permissions, device growth?, Apple Maps updates "*. July 2014. URL: `http://www.theguardian.com/technology/blog/2014/jul/08/android-permissions-apple-maps`.

[17] David Nadeau and Satoshi Sekine. "A survey of named entity recognition and classification". In: *Linguisticae Investigationes* 30.1 (2007), pp. 3–26.

[18] Dragos Sbirlea et al. "Automatic detection of inter-application permission leaks in Android applications". In: *IBM Journal of Research and Development* 57.6 (2013).

[19] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32.

[20] Android Developer Program. *"Activity Manager API"*. URL: `http://developer.android.com/reference/android/app/ActivityManager.html`.

[21] Android Team. *"Provide Broadcast or API to Get Current Foreground Activity"*. URL: `https://code.google.com/p/android-developer-preview/issues/detail?id=29#c50`.

[22] Matt Warman. *"Facial recognition: inevitable, but will shoppers approve?"* Nov. 2013. URL: `http://www.telegraph.co.uk/technology/news/10435521/Facial-recognition-inevitable-but-will-shoppers-approve.html`.

[23] Kelsey Libert. *"Age and Gender Matter in Viral Marketing"*. Aug. 2014. URL: `http://selfiecity.net`.

[24] Jiaqi Tan et al. "Short Paper: CHIPS: Content-based Heuristics for Improving Photo Privacy for Smartphones". In: *Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless & Mobile Networks*. WiSec '14. Oxford, United Kingdom, 2014.