What Data Do The Google Dialer and Messages Apps On Android Send to Google?

Douglas J. Leith
Trinity College Dublin, Ireland
28th Feb 2022

***Abstract*—We report on measurements of the data sent to Google by the Google Messages and Google Dialer apps on an Android handset. We find that these apps tell Google when message/phone calls are made/received. The data sent by Google Messages includes a hash of the message text, allowing linking of sender and receiver in a message exchange. The data sent by Google Dialer includes the call time and duration, again allowing linking of the two handsets engaged in a phone call. Phone numbers are also sent to Google. In addition, the timing and duration of other user interactions with the apps are sent to Google. There is no opt out from this data collection. The data is sent via two channels, (i) the Google Play Services Clearcut logger and (ii) Google/Firebase Analytics. This study is therefore one of the first to cast light on the actual telemetry data sent by Google Play Services, which to date has largely been opaque. We informed Google of our findings and delayed publication for several months to engage with them. On foot of this report Google say that they plan to make multiple changes to their Messages and Dialer apps.**

I. Introduction

We analyse the data sent to Google by Android handsets using the Google Messages and Google Dialer apps. Both are core apps for a mobile handset, the Messages app being used to send and receive SMS text messages and the Dialer app to make/receive phone calls. According to the Google Play store the Google Messages app is installed on >1 Billion handsets. In the US, AT&T and T-Mobile recently announced all Android phones on their networks will use the Google Messages app[1] and the app also comes pre-loaded on recent Samsung handsets[2] and on Xiaomi and Huawei handsets. According to the Google Play store the Google Dialer app is also installed on >1 Billion handsets.

In summary, we find that:

1) When an SMS message is sent/received the Google Messages app sends a message to Google servers recording this event, the time when the message was sent/received and a truncated SHA256 hash of the message text. The latter hash acts to uniquely identify the text message. The message sender's phone number is also sent to Google, so by combining data from handsets exchanging messages the phone numbers of both are revealed.
2) When a phone call is made/received the Google Dialer app similarly logs this event to Google servers together with the time and the call duration.

This data is sufficient to allow discovery of whether a pair of handsets are communicating.

The data sent to Google is tagged with the handset Android ID, which is linked to the handset's Google user account and so often to the real identity of the person involved in a phone call or SMS message. For example, a working phone number is required to create a Google account, and if the person has paid for an app on the Google Play store or uses Google Pay then their Google account is also linked to their credit card/bank details. In this way real-world identities of the pair of people communicating may be revealed to Google.

In addition to logging the sending/receiving of SMS messages and phone calls, the Google Messages and Dialer apps send messages to Google recording user interactions with the app. For example, when the user views an app screen, an SMS conversation or searches their contacts the nature and timing of this interaction is sent to Google allowing a detailed picture of app usage over time to be reconstructed.

There is no opt out from this data collection.

When, in addition, the "See caller and spam ID" option is enabled in the Google Dialer app (which is the default) the app sends the phone number of each incoming call[3] to Google, together with the time of the call. By combining data from handsets exchanging phone calls the phone numbers of both are therefore revealed. We note that sending of incoming phone numbers to Google is *not* necessary for call screening. For example, Google's Safe Browsing anti-phishing service for web browsers achieves URL screening while only uploading partial URL hashes to Google servers (these partial hashes are used to download a list of blacklisted sites which can then be compared locally against the URL concerned) [1], [2].

The Google Messages and Dialer apps send data to Google via two channels: (i) the Google Play Services Clearcut logger service and (ii) Google/Firebase Analytics. Recent Android measurement studies have noted the large volume of data sent by Google Play Services to Google servers on most Android

[1] https://www.theverge.com/2021/6/30/22556686/att-android-phones-rcs-google-messages

[2] https://support.google.com/messages/answer/10324785?hl=en

[3] Google state that only incoming phone numbers that are not saved in a user's contacts are sent to them. We note that on our test phones the contacts list was empty.

handsets [3], [4]. A substantial component of this data is sent by the Clearcut logger service within Google Play Services. However, the data transmission is largely opaque, being binary encoded with little public documentation [3], [4].

The Google Play Services support page[4] states that data is collected for (i) security and fraud prevention, (ii) to provide, maintain and improve Google Play Services APIs and core services and (iii) to provide Google services such as syncing of bookmarks and contacts. However, few details are given as to the actual data collected. Google have also publicly stated that Google Play Services data is "essential for core device services such as push notifications and software updates across a diverse ecosystem of devices and software builds"[5].

The work reported here is the first close look at the actual data sent by the Clearcut logger component of Google Play Services. It is limited in nature – we focus only on the data that the Messages and Dialer apps send via Google Play Services. This is due to the time-consuming nature, in the absence of public documentation, of the work involved in decoding the binary data sent by Google Play Services. Nevertheless, our measurements are already enough to establish that the data sent goes beyond what is suggested by the Google Play Services support page and Google's public statements. The data sent is not simply system health data (battery and CPU statistics and the like), device configuration data needed to check for updates, syncing of contacts and account details etc, but rather extends to details of the phone calls and SMS messages sent/received by users, and of user interactions with the Messages and Dialer apps (which SMS conversations viewed and when, dialing of phone numbers and so on).

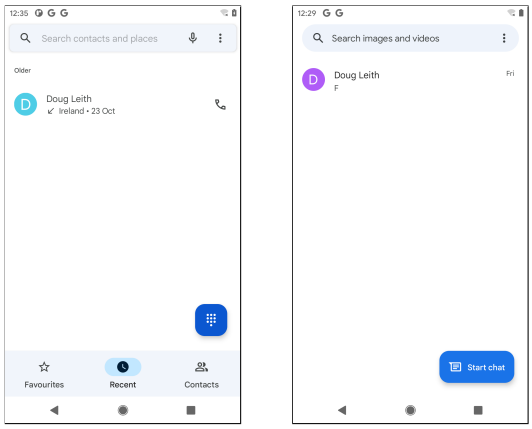
We note that we made a request using Google's https://takeout.google.com/ portal for the data associated with the Google user account used in our tests. The response to this request did not include the call/SMS and user interaction log data that we observed to be collected.

While we report here on Android 11 measurements, we observed the same behaviour on a Pixel 4a handset running Android 12.

It is important to emphasise that the measurements reported here are certainly not comprehensive, even for the Messages and Dialer apps. We did not look at all of the log sources within these apps that send data to the Google Play Services Clearcut logger, nor at cascade events such as logging generated by components of Google Play Services itself triggered by typing in the app search bar.

A. Mitigations

1) How To Find Which Dialer and Message Apps Are Installed: Unfortunately, it is not that straightforward to determine whether the dialer and message apps installed on your handset are from Google. One quick check is to open the



(a) Google Dialer (b) Google Messages

Fig. 1: Google Dialer and Messages app home screens

installed apps and compare with the screenshots in Figure 1, but this may be unreliable. Probably the simplest, safest way to reliably check is to install the APK Explorer app from the F-Droid app store[6]. This is a verified open source app without embedded trackers. Opening the app displays a list of installed apps and their unique package names. The package name of the Google Dialer is com.google.android.dialer and of Google Messages is com.google.android.app.messaging, so search for these.

Handset brands that come with Google Messages pre-installed include Xiaomi, Huawei, Google and newer Samsung and One Plus phones. Handset brands that come with the Google Dialer pre-installed include Xiaomi, Google and newer One Plus phones. AT&T and T-Mobile recently announced that all Android phones on their networks will in future use the Google Messages app[7].

2) Tracker-Free Alternatives: On Android it is possible to change the default dialer and messages apps. Verified open-source, tracker-free dialer and messages apps are available on the F-Droid app store. For example, Simple Dialer[8], QKSMS[9] and Simple SMS Messenger[10].

B. GDPR

We report on a technical study here, not a legal one, and in any case we are not legally qualified. Nevertheless, the data collection that we observe by Google raises obvious questions regarding GDPR data protection regulations in Europe (the measurements were all carried out within Europe using handsets purchased in Europe). Roughly speaking, there are three main basis under GDPR for data collection[11]: (i) the data is anonymised, i.e. cannot reasonably be linked to an individual person, and so is not personal data, (ii) with consent for a defined purpose and (iii) for the legitimate interests of Google.

[4]https://support.google.com/android/answer/10546414?hl=en

[5]E.g. see https://www.bleepingcomputer.com/news/security/study-reveals-android-phones-constantly-snoop-on-their-users/

[6]https://f-droid.org/en/packages/com.apk.editor/

[7]https://www.theverge.com/2021/6/30/22556686/att-android-phones-rcs-google-messages

[8]https://f-droid.org/en/packages/com.simplemobiletools.dialer/

[9]https://f-droid.org/en/packages/com.moez.QKSMS/

[10]https://f-droid.org/en/packages/com.simplemobiletools.smsmessenger/

[11]E.g. see https://gdpr.eu/what-is-gdpr/

1) Lack of Anonymity: Regarding anonymity, all of the events recorded via the Google Play Services Clearcut logger are tagged with the handset's Android ID. Via other data collected by Google Play Services this ID is linked to (i) the handset hardware serial number, (ii) the SIM IMEI (which uniquely identifies the SIM slot) and (iii) the user's Google account. When a SIM is inserted the Google Messages app also links the Android ID to the SIM serial number/ICCID, which uniquely identifies the SIM card.

By making a request using https://takeout.google.com/ for the data associated with the Google user account used in our tests we further confirmed that the data reported under the heading "Android Device Configuration Service" includes the Android ID for each handset used (as well as the handset serial number, SIM IMEI, last IP address used and mobile operator details).

When creating a Google account it is necessary to supply a phone number on which a verification text can be received. For many people this will be their own phone number. Use of Google services such as buying a paid app on the Google Play store or using Google Pay further links a person's Google account to their credit card/bank details. A user's Google account, and so the Android ID, can therefore commonly be expected to be linked to the person's real identity.

Additionally, when a message is received by the Google Messages app the sender's phone number is sent to Google via the Google Play Services Clearcut logger, see Section V-B2. By combining data from the pair of handsets involved in an exchange of messages (which seems perfectly feasible based on the hashes of the message text that we observe to be sent to Google) both phone numbers may be revealed and linked to the Android IDs. Similarly when the spam protection option is enabled in the Google Dialer (as it is by default), see Section VI-A4.

All of the events recorded via Google Analytics are tagged with the user's Google Advertising ID and the sender app's Firebase ID. The app Firebase ID is directly linked to the handset Android ID when the app registers to use the Google Analytics service, see Section III-E1.

The linkage between the various identifiers is illustrated schematically on Figure 2.

2) No Consent: Specific consent has neither been sought nor given for the data collection by the Google Messages and Dialer apps that we observe, and there is no opt out.

3) Legitimate Interest: Invoking legitimate interest requires the data to be collected for a specific purpose, that the data is necessary for the purpose, that the data collection is balanced against the interests and freedoms of the individual, and so on[12]. The legitimate interest basis for data collection is the least clear, and probably best left to the lawyers. We note, however, that we could not find an app-specific privacy policy stating the specific purpose for which the data that we observe is collected and the basis used for data collection. We discuss this further next.

[12]E.g. see https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/lawful-basis-for-processing/legitimate-interests/.

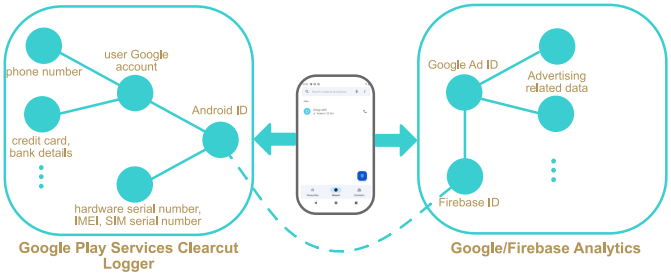


Fig. 2: Ilustrating how handset data can be linked to a person's real identity. Handset data sent to Google via the Google Play Services Clearcut logger is tagged with the Android ID, which in turn is linked to the user's Google account and to device/SIM identifiers. The user's Google account in turn may be connected to the person's phone number, credit card/bank details etc and so their real identity. Handset data sent to Google via Google/Firebase Analytics is tagged with the Google Advertising ID and the Firebase ID of the app carrying out the data collection. The Google Advertising ID links this data with other data collected for advertising-related purposes. The Firebase ID is linked to the Android ID, and so to the user Google account etc.

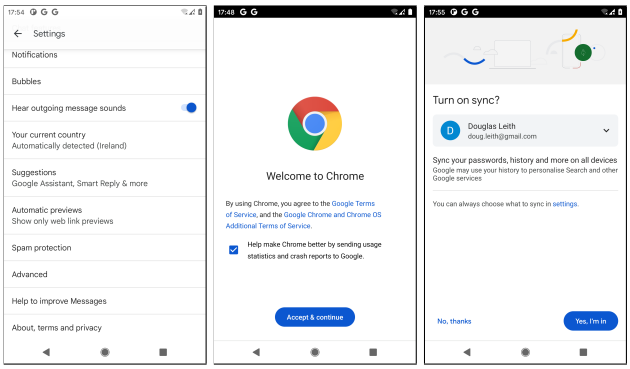
C. Lack of App-Specific Privacy Policy

1) Google Messages: Viewing the privacy policy of the Google Messages app is not straightforward. It is necessary to: (i) click on the three dots in search bar to open the Settings menu, (ii) scroll down to see an "About, terms and privacy" link (see Figure 3(a)), (iii) click on this to open a new menu that shows a "Privacy Policy" link, (iv) click on this link which opens a Google Chrome window. At this point, to proceed it is necessary to agree to the Google Chrome terms and conditions, see Figure 3(b). It is not possible to proceed to view the Messages app privacy policy without first agreeing to the additional Google Chrome terms and conditions.

This process of navigating multiple menus and links hardly seems best practice. Mandating acceptance of Google Chrome terms and conditions in order to view the Messages app privacy policy is poor practice.

We note also that the Messages app silently sends messages to Google via Google Analytics logging the fact that the page with the privacy policy link has been viewed, see Section V-D. This occurs before the privacy policy itself has been viewed.

Agreeing to the Google Chrome terms and conditions leads to a page encouraging use of Google's sync service, see Figure 3(c) and after passing through that the user is finally allowed to view the privacy policy web page at http://www.google.com/intl/en_IE/policies/privacy/ (note the use of http rather than https, although this redirects to https://policies.google.com/privacy?hl=en&gl=IE). Unfortunately, this is not an app-specific privacy policy but the rather general Google privacy



(a) (b) (c)

Fig. 3: Navigating to the Google Messages app privacy policy page requires navigating multiple menus and links and agreeing to the additional Google Chrome terms and conditions.

policy. This is silent on the specific data collected by the Messages app, the associated app-specific purposes and the basis under which this app-specific data is collected.

We note that during the loading of this privacy policy web page around 20 connections are made that appear to send telemetry to Google servers, see Section V-D.

2) Google Dialer: The Google Dialer does not appear to have an app privacy policy link, only a privacy policy associated with the Support pages. The only privacy policy link that we could find was by following these steps: (i) click on the three dots in search bar to open a menu, (ii) click on the "Help and feedback" to open the Support page, (iii) click on the three dots at the top right of the Support page to open a new menu, (iv) click on the "Privacy Policy" link that is revealed. As with the Google Messages app this process opens a Google Chrome window and it is necessary to agree to the additional Google Chrome terms and conditions in order to proceed. The user eventually arrives at the web page http://www.google.com//policies/privacy/ which redirects to https://policies.google.com/privacy. This is Google's global privacy policy page, with no app specific information and not localised to Europe/Ireland (unlike for the Messages app). Similarly to the Messages app, loading the privacy policy page prompts multiple connections including to www.youtube-nocookie.com/youtubei/v1/log_event sending what appears to be telemetry, and to www.google-analytics.com/j/collect, stats.g.doubleclick.net/j/collect.

D. Recommendations to Google

In light of our observations we make the following recommendations (in no particular order):

1) The specific data collected by Dialer and Messages apps, and the specific purposes for which it is collected, should to be clearly stated in the app privacy policies.

2) The app privacy policy should be easily accessible to users and be viewable without having to first agree to other terms and conditions (e.g. those of Google Chrome). Viewing of the privacy policy should not be logged/tracked prior to consent to data collection.

3) Data on user interactions with an app, e.g. app screens viewed, buttons/links clicked, actions such as sending/receiving/viewing messages and phone calls, is different in kind from app telemetry such as battery usage, memory usage, slow operation of the UI. User's should be able to opt out of collection of their interaction data.

4) User interaction data collected by Google should be made available to users on Google's https://takeout.google.com/portal (where other data associated with a user's Google account can already be downloaded).

5) When collecting app telemetry such as battery usage, memory usage etc, the data should only be tagged with short-lived session identifiers, not long-lived persistent device/user identifiers such as the Android ID

6) When collecting data only coarse time stamps should be used, e.g. rounded to the nearest hour. The current approach of using timestamps with millisecond accuracy risks being too revealing. Better still, use histogram data rather than timestamped event data, e.g. a histogram of the network connection time when initiating a phone call seems sufficient to detect network issues.

7) Halt the collection of the sender phone number via the `CARRIER_SERVICES` log source when a message is received, and halt collection of the SIM ICCID by Google Messages when a SIM is inserted. Halt collection of a hash of sent/receivedmessage text.

8) The current spam detection/protection service transmits incoming phone numbers to Google servers. This should be replaced by a more privacy-preserving approach, e.g. one similar to that used by Google's Safe Browsing anti-phishing service which only uploads partial hashes to Google servers [1], [2].

9) A user's choice to opt out of "Usage and diagnostics" data collection should be fully respected i.e. result in a halt to all collection of app usage and telemetry data.

E. Response From Google

The apps studied here are in active use by many millions of people. We informed Google of our findings, delayed publication to allow them to respond and in fairness to Google they have engaged positively with us. In summary,

1) Google say they plan to change the app onboarding flow so that users are notified this is a Google app with a link to Google's consumer privacy policy. This will likely include opportunities to provide more "Privacy Tours" that walk the user through an overview of the app's data use and data collection. This will include a new on/off toggle to cover data collection that Google do not consider to be essential for the app to function.

2) Will halt the collection of the sender phone number via the `CARRIER_SERVICES` log source, collection of the

SIM ICCID and of a hash of sent/receivedmessage text by Google Messages (the latter change will be rolled out with version 10.9.160 of Google Messages, the other changes in the next release).

3) Will remove logging of call related events in Firebase Analytics from both Google Dialer and Messages.

4) Re the recommendation to use short-lived session identifiers for telemetry data, Google say they would like to see more logging moved to using the least long-lived identifier available whenever possible and that this an ongoing project.

5) Re the spam detection/protection service, Google note that this only occurs for phone numbers not in the handset contacts list and plan to (i) create a product tour explaining to new users and reminding current users that caller ID and spam protection is turned on for user protection, and letting them know how to disable it, (ii) add a visual indicator within the Messages app that indicates when spam protection is enabled, (iii) investigate whether an approach similar to the Safe Browsing hash prefix solution can be used. Google also state that the timestamp logged in the SCOOBY_EVENTS log message (see Section VI.A.4) is fuzzed to the nearest hour server-side, and will also be fuzzed client-side from version v75 onwards of the Dialer app.

6) Google state that there are back-end server controls to regulate joins between the Android ID and user account data, but the policy used to manage joins is not publicly available. Google also note that when a handset has multiple Google user accounts then its Android ID would be associated with all of those user accounts.

It's worth noting that this summary of our discussions is written by us, not Google. It reflects our understanding of those discussions but any mistakes are of course our own.

Google also provided clarification on the purposes of some of the data collection observed. Namely:

1) The message hash is collected for detecting message sequencing bugs.

2) Phone numbers are collected to improve regex pattern matching for automatic recognition of one-time passwords sent over RCS. Messages automatically recognizes incoming One-Time Password (OTP) codes to avoid the user having to fill them in. This can be a frequent point of failure and the phone number data is used to improve recognition by providing ground-truth based on known OTP sender numbers.

3) The ICCID data is used to support Google Fi.

4) Firebase Analytics logging of events (not including phone numbers) is used to measure the effectiveness of app download promotions (for Messages and Dialer specifically). Namely, to measure not only whether the app was downloaded but also whether it was used once downloaded.

II. Related Work

Probably closest to the present work are recent analyses of the data shared by Google Play Services [5], [3], [4]. The measurement study in [5] was motivated by the emergence of Covid contact tracing apps based on the Google-Apple Exposure Notification (GAEN) system, which on Android requires that Google Play Services to be enabled. This highlighted the extensive data collection Google Play Services. The follow-up work in [3] extended consideration to the data sent to Apple by an iPhone/IOS. Recently, in [4] the data sent by six variants of the Android OS, namely those developed by Samsung, Xiaomi, Huawei, Realme, LineageOS and e/OS, is measured (in [5], [3] only Google-brand Android handsets were studied). While the focus was on data sent to non-Google servers, e.g. on the data sent to Samsung by a Samsung-brand handset, this study again highlighted the large volume of data uploaded to Google by Google Play Services on all handsets apart from the e/OS handset. The volume of data uploaded to Google was observed to be at least $10\times$ that uploaded by the mobile OS developer, rising to around $30\times$ for the Xiaomi, Huawei and Realme handsets. This occurs despite the 'usage & diagnostics" option being disabled for Google Play Services in these studies. These previous studies also note the opaque nature of this data collection by Google, with there being no public documentation, use of binary encoded payloads and obfuscated code.

The microG project[13] is an open source re-implementation of parts of the Google Play Services API used by popular apps (in particular the Fused Location, Maps, Firebase Cloud Messaging/push notifications, authentication and SafetyNet components). However, the microG project has specifically avoided re-implementation of the analytics components of Google Play Services, including Google/Firebase Analytics and the Clearcut logger service, and it is these that we study here.

III. The Challenge of Seeing What Data Is Sent

It is generally straightforward to observe packets sent from a mobile handset. Specifically, we configure the handsets studied to use a WiFi connection to a controlled access point, on which we use `tcpdump` to capture outgoing traffic. However, this is of little use for privacy analysis because: *(i)* packet payloads are almost always encrypted due to the widespread use of HTTPS to transfer data; *(ii)* prior to message encryption, data is often encoded in a binary format for which there is little or no public documentation

A. Decrypting HTTPS Connections

Almost all of the data we observe is sent over HTTPS connections and so encrypted using TLS/SSL (in addition to any other encryption used by the app). However, decrypting SSL connections is relatively straightforward. We route handset traffic via a WiFi access point (AP) that we control, configure this AP to use `mitmdump` as a proxy [6] and adjust the

[13]https://microg.org/.

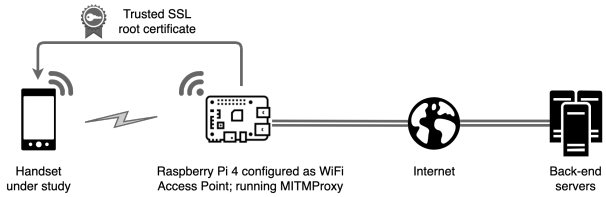


Fig. 4: Measurement setup. Mobile handset configured to access the Internet using a WiFi access point hosted on a Raspberry Pi. A system certificate is installed on the phone to be able to decrypt outgoing traffic. The accesspoint pretends to any process running on the handset to be the destination server, creates a connection to the actual target, and relays requests and their replies between handset and server while logging the traffic.

firewall settings to redirect all WiFi HTTP/HTTPS traffic to mitmdump so that the proxying is transparent to the handset. When a process running on the handset starts a new network connection, the mitmdump proxy pretends to be the destination server and presents a fake certificate for the target server. This allows mitmdump to decrypt the traffic. It then creates an onward connection to the actual target server and acts as an intermediary, relaying requests and their replies between the app and the target server while logging the traffic. The setup is illustrated schematically in Figure 4.

System processes typically carry out checks on the authenticity of server certificates received when starting a new connection and abort the connection when these checks fail. For Google apps and services, installing the mitmproxy CA cert as a trusted certificate causes these checks to pass. Installing a trusted cert is slightly complicated in Android 10 and later, since the system disk partition, on which trusted certs are stored, is read-only and security measures prevent it being mounted as read-write. Fortunately, folders within the system disk partition can be overriden by creating a new mount point corresponding to the folder, and in this way the mitmdump CA cert can be added to the `/system/etc/security/cacerts` folder.

B. Google Play Services Telemetry

The Google Message and Dialer apps do not send data directly to Google, but rather send data to event logging services within Google Play Services. Specifically, to the Clearcut logger service and the Google/Firebase Analytics service. These Google Play Service components expose APIs that the app uses to communicate with them. The Clearcut logger and Google/Firebase Analytics services then batch up data received and forward it to Google servers. The Clearcut logger sends data to https://play.googleapis.com/log/batch while Google/Firebase Analytics sends data to https://app-measurement.com/. This process is illustrated schematically in Figure 5.

The Clearcut logger and Google/Firebase Analytics services encode the data in different formats for sending to Google. We discuss these formats in more detail next.

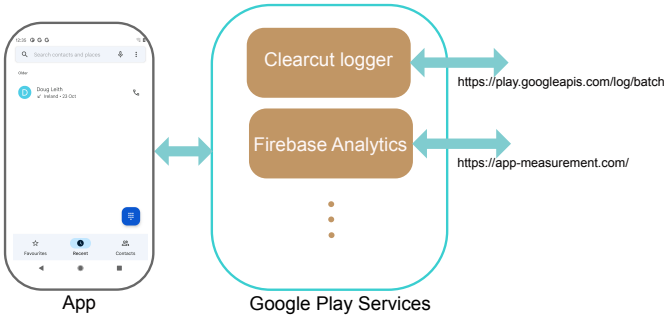


Fig. 5: Schematic illustrating app data flow. The app sends event data to Google Play Services via the Clearcut logger and Firebase Analytics APIs. These Google Play Services components then batch up the data and send it to Google servers. Note that Google Play Services provides many other APIs and services in addition to the Clearcut logger and Firebase Analytics.

C. Decoding Google Play Services Clearcut Logger Data

The Clearcut logger service within Google Play Services sends data to https://play.googleapis.com/log/batch. Each message sent includes an NID cookie and an x-server-token authentication token (which act as device identifiers), followed by the message body, e.g.

```
POST https://play.googleapis.com/log/batch
Headers
   x-server-token: CAESOQDyi0h8...YWXxG0vLQ
   cookie: NID=511=Oy6F1KJ7JeZ...yZ0RhdX8o6efg
<Body>
```

The message body is encoded in a binary protobuf format[14]. Figure 6 shows the structure of the decoded message, including an example header message, and Table I gives a list of log sources observed by [3] (this list is probably not comprehensive). Note that the sequence of log entries sent by each log source is encoded as a protobuf array. That is, as a sequence of <length/varint><protbuf> entries from which the individual log entry protobufs need to be extracted and decoded. Standard tools cannot decode a protobuf array but we have made software tools that we have developed for this publicly available, see below.

Protobufs can be decoded without knowledge of the message content using the Google Protobuf compiler with the `--decode_raw` option. However, this means that the interpretation of values is missing and there is also sometimes ambiguity as to interpretation of the value types. Figure 7(a) shows an example of a log entry generated by the Google Messages app `ANDROID_MESSAGING` log source and decoded in this way. While the contents of the log entry can be viewed, it remains largely opaque since the interpretation of the various numerical and string values is not known.

Since there is no public documentation, to determine the meaning of these values we (i) decompile the Google Messages app, (ii) identify the protobuf used to encode the log entry within the decompiled code (this step is non-trivial since the Google

[14]https://developers.google.com/protocol-buffers/

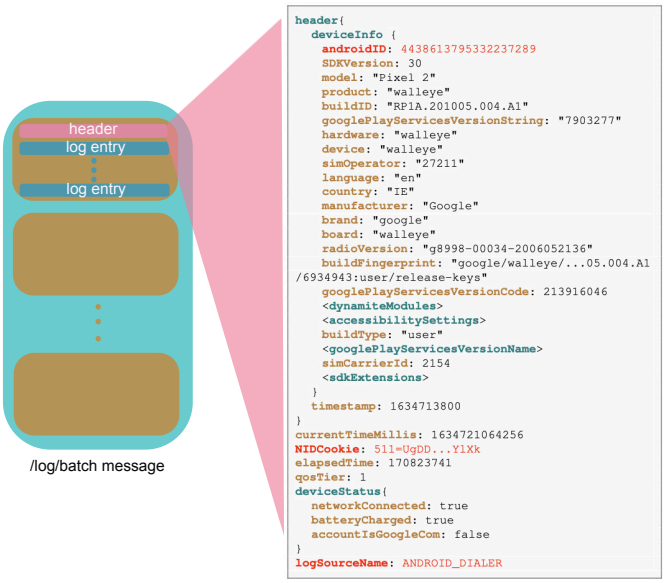


Fig. 6: Structure of messages sent to play.googleapis.com/log/batch by the Google Play Services Clearcut logger. Each message consists of one or more bundles of log entries, indicated in brown. Each bundle has a header containing device details and persistent identifiers (Google androidID, NID cookie) and specifying the log source. This header is followed by one or more log entries, the format of the log entries being determined by the log source.

CARRIER_SERVICES, **ANDROID_DIALER**, ONEGOOGLE_MOBILE, GOOGLE_NOW_LAUNCHER, DRIVE, COPRESENCE_NO_IDS, AUTOFILL_WITH_GOOGLE, **SCOOBY_EVENT**, SCOOBY_EVENT_LOG, BEACON_GCORE, NETREC, BRELLA, GOOGLE_HELP, PHOTOS, CALENDAR, CALENDAR_UNIFIED_SYNC, BUSINESS_VOICE, IDENTITY_FRONTEND, GMS_CORE_PEOPLE, LATIN_IME, DL_FONTS, CAR, ICING, ACTIVITY_RECOGNITION, ANDROID_CONTACTS, ANDROID_GROWTH, ANDROID_GSA, CLIENT_LOGGING_PROD, GOOGLETTS, CAST_SENDER_SDK, ANDROID_VERIFY_APPS, ANDROID_BACKUP, **ANDROID_MESSAGING**, ANDROID_OTA, ANDROID_GMAIL, ANDROID_SNET_GCORE, GAL_PROVIDER, GLAS, TACHYON_LOG_REQUEST, CLEARCUT_FUNNEL, CLEARCUT_LOG_LOSS, DIALER_ANDROID_PRIMES, CARRIER_SERVICES_ANDROID_PRIMES, TURBO_ANDROID_PRIMES, PHOTOS_ANDROID_PRIMES, ANDROID_MESSAGING_PRIMES, GOOGLETTS_ANDROID_PRIMES, SETTINGS_INTELLIGENCE_ANDROID_PRIMES, ANDROID_GSA_ANDROID_PRIMES, SAFETYHUB_ANDROID_PRIMES, WIFI_ASSISTANT_PRIMES, DRIVE_ANDROID_PRIMES, GMAIL_ANDROID_PRIMES, STREAMZ_ANDROID_GROWTH, STREAMZ_ANDROID_GSA, STREAMZ_ONEGOOGLE_ANDROID, STREAMZ_HERREVAD, STREAMZ_CALENDAR, STREAMZ_PHOTOS_ANDROID, STREAMZ_ANDROID_AUTH_ACCOUNT, STREAMZ_GELLER, STREAMZ_NGA, BUGLE_COUNTERS, PSEUDONYMOUS_ID_COUNTERS, GMAIL_COUNTERS, WESTWORLD_COUNTERS, GOOGLE_KEYBOARD_COUNTERS, ANDROID_CONTACTS_COUNTERS, WALLPAPER_PICKER_COUNTERS, PLATFORM_STATS_COUNTERS

[

TABLE I: Log source names observed in Google Play Services Clearcut logger messages sent to play.googleapis.com/log/batch (taken from [3]). The log sources studied here are highlighted in red.

Messages app contains more than 2000 distinct protobufs[15].) and then (iii) trace back within the code to determine how the value of each entry in the protobuf is calculated. Figure 7(b) shows the result of this fairly laborious process.

It can be seen that many of the numerical values within the message encode event and state information, for example the number 2 in field 1 encodes the fact that this is a BUGLE_MESSAGE event (Bugle is the internal name used for the Messages app). Note that the enum labels here, e.g. BUGLE_MESSAGE, are extracted from the app code and so are Google's own. Other values encode the fact that a message was successfully sent as part of a conversation (`bugleMessageSource`) as well as the time (in milliseconds since 1st Jan 1970) when the message was sent (`currentTime_ms`) and the log entry was sent (`timestamp`). Observe also the two truncated SHA256 hash values near the bottom of the message. We shall return to these shortly, but note that the `sha256HashMsg` value is a hash of the time, in hours since 1st Jan 1970, that the message was sent and of the message content i.e the message text, truncated to 128 bits. This hash uniquely identifies the message that was sent. The `sha256HashPrevMsg` similarly identifies the previous message sent/received in the conversation. In the absence of documentation the interpretation of such values is simply impossible without the kind of time-consuming reverse engineering carried out here.

(a) (b)

Fig. 7: Example of Google Messages `ANDROID_MESSAGING` Clearcut logger log entry: (a) protobuf decoded using Google Protobuf compiler with the `--decode_raw` option, (b) after reverse engineering the schema.

Each log source sending data to the Clearcut logger service uses its own protobuf format for log entries, necessitating seperate reverse engineering of each in order to decode the message content.

Figure 8 shows an example of a decoded Google Dialer log entry generated in response to manually dialing a phone number. The `AOSPEventType` value `MAIN_CLICK_FAB_TO_OPEN_DIALPAD` records the fact that the dialpad was opened, and the `timestamp` records the time when this occurred. As the phone number is typed a `searchQuery` event is logged for each digit typed, see Figure 8(a) for an example log entry sent

[15]The protobufs themselves are encoded within the app in compact protobuf format, which is undocumented although there are useful comments embedded in the Android source code, see https://cs.android.com/android/platform/superproject/+/master:external/protobuf/java/core/src/main/java/com/google/protobuf/RawMessageInfo.java

```
logEntry:{
  timestamp: 1635013503410
  event {
    impressionEvent {
      timestamp: 1635013503410
      deviceDetails {
        buildDevice: "walleye"
        buildModel: "Pixel 2"
        buildVersionRelease: "11"
        buildID: "RP1A.201005.004.A1"
        releaseStatus: RELEASE
        systemApp: true
        updatedSystemApp: true
        deviceClass: DEFAULT_GOOGLE_DEVICE
        simOperator: "Tesco Mobile"
        country: "ie"
        elapsedDaysSinceDialerInstall: 4678
        mobileOperator: "27205"
        installedBy: "com.android.vending"
        14: "0AFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"
      }
      AOSPEventType: MAIN_CLICK_FAB_TO_OPEN_DIALPAD
    }
  }
  subEvent: 1340
  tz_offset: 3600
  <...>
}
```

Fig. 8: Example of Google Dialer `ANDROID_DIALER` Impression event log entry recording event that dialpad has been opened.

```
logEntry:{
  timestamp: 1635013208570
  event {
    searchQuery {
      queryLength: 3
      searchTimeMillis: 47
      5: 1
    }
  }
  tz_offset: 3600
  <...>
}
```

(a)

```
callDetails {
  isIncoming: False
  callInitiationType: Dialpad
  lookupResultType: NOT_FOUND
  disconnectCause: REMOTE
  callDuration: 48605 // ms
  callSetupTiming {<...>}
  deviceDetails {<...>}
  callID: "d97ae1a6-...7d7ecb7"
  <...>
}
```

(b)

Fig. 9: More examples of Google Dialer `ANDROID_DIALER` log entries: (a) logging each key press when dialing a phone number, (b) call details sent upon completion of a call, including the call duration (in milliseconds).

in response to a keypress. When a phone call finishes this event is also logged, using a message similar to that in Figure 8 but with `AOSPEventType` value `USER_PARTICIPATED_IN_A_CALL` and including additional data recording call details, including the call duration (in milliseconds), see Figure 8(b).

D. Decoding Google Play Services /checkin Message

Google Play Services sends periodic messages to android.googleapis.com/checkin that act to link together a number of persistent device and user identifiers, see [7], [5], [3], [4]. An abridged decoded example of one these connections is as follows:

```
POST https://android.googleapis.com/checkin
Headers:
  Cookie:
NID=511=H_JOLe0z_gYPWW...oo_n5wHitQ
Body:
  imei: "357537089248629"
  androidId: 4438613795332237289
  <...>
  IMEI: "35753708924862"
  account "[doug.leith@gmail.com]"
  accountToken: "ya29.a0AR...mUWA"
  timeZone: "Europe/Dublin"
  securityToken: 8517022668257493554
  <...>
  hardwareSerialNumber: "HT85G1A05551"
  <...>
```

Observe that the message contains the (i) Android ID (a long-lived device identifier that can only be changed by carrying out a factory reset), (ii) the IMEI (which uniquely identifies the handset SIM slot), (ii) the hardware serial number (which uniquely identifies the handset), (iv) the NID cookie (which acts as a persistent device identifier), (v) the Google account username/email (which identifies the handset user) and (vi) a user account authorisation token (which again identifies the handset user).

As already noted, logging messages sent by the Clearcut logger to https://play.googleapis.com/log/batch are tagged with the AndroidID, and so via this /checkin message can be linked to long-lived device and user identifiers.

E. Decoding Google/Firebase Analytics Event Logging

In addition to logging events via the Google Play Services Clearcut logger service, the Google Dialer and Messages apps also log events using the Google/Firebase Analytics service. Decoding of Google Analytics messages is much simpler than for the Clearcut logger service since the same protobuf format is used for all messages.

1) Registering App With Google/Firebase Analytics: Apps using Google/Firebase Analytics first register with the service by connecting to android.clients.google.com/c2dm/register3. For example, here is the Google Dialer registering:

```
POST https://android.clients.google.com/c2dm/register3
Headers:
  Authorization: AidLogin 4438613795332237289:8517022668257493554
  app: com.google.android.dialer
Body:
X-subtype=566865154279&sender=566865154279&X-app_ver=7903277&X-osv=30&X-cliv=
fiid-21.1.1&X-gmsv=213916046&X-appid=f86VDMH_SSGcArM16Up973&X-scope=*&
X-Goog-Firebase-Installations-Auth=eyJhb...UiZvYtC&X-gmp_app_id
=1:566865154279:android:85888a731cba65c4ed1242&X-Firebase-Client=
fire-analytics/19.0.2+fire-core/19.3.2_lp+fire-fcm/20.1.7_lp+fire-android/+
fire-installations/16.3.6_lp+fire-iid/21.1.1&X-firebase-app-name-hash=R1dAH9
...1tLxhI&X-Firebase-Client-Log-Type=1&X-app_ver_name=70.05.401408800&app=com.
google.android.dialer&device=4438613795332237289&app_ver=7903277&info=s_uTu6
...yRc&gcm_ver=213916046&plat=0&cert=203...a0439&target_ver=30
<<< HTTP 200, 169.00B
```

The `AidLogin` header contains the AndroidID value 4438613795332237289 plus a security token. The `X-appid` value is the Firebase ID, which uniquely identifies the Google Dialer app instance. This Firebase ID is also encoded in the `X-Goog-Firebase-Installations-Auth` value which is a JWT token[16] that decodes to:

```
{
  "appId": "1:566865154279:android:85888a731cba65c4ed1242",
  "exp": 1635322812,
  "fid": "f86VDMH_SSGcArM16Up973",
  "projectNumber": 566...54279
}
```

2) Google/Firebase Analytics Event Logging: After registering with Google/Firebase Analytics, apps can log events. Logged events are batched and sent to https://app-measurement.com/a. Each event is tagged with app, device and user information, see Figure 10. This includes the app Firebase ID (which uniquely identifies the app instance is linked to the handset via the AndroidID during the registration with Google/Firebase Analytics, see above) and the handset Google Advertising ID (which links the event with other user data collected for advertising purposes), see Section VII for further discussion of this.

[16]https://jwt.io/

```
bundle {
  <event>
  <user_info>
  message_timestamp: 1635013256591
  event_timestamp: 1635013211681
  bundle_end_timestamp: 1635013250642
  last_bundle_end_timestamp: 1635013039074
  operating_system: "android"
  operating_system_version: "11"
  Build_MODEL: "Pixel 2"
  language_country: "en-ie"
  timezone_offset_mins: 60
  package_name: "com.google.android.dialer"
  app_version: "70.05.401408800"
  gmp_version: 44007
  gms_version: 213916
  google_ad_id: "916c714a-e838-479d-a7a6-3325d838da5f"
  random_hex: "27119320961e981207e9f885ddfb897b"
  dev_cert_hash: 12863297414841520258
  daily_conversions_count: 17
  gmp_app_id: "1:566865154279:android:85888a731cba65c4ed1242"
  last_bundle_start_timestamp2: 1635013035855
  firebase_instance_id: "f86VDMH_SSGcArM16Up973"
  app_version_int: 7903277
  config_version: 1631224292436546
  dynamite_version: 53
}
```

Fig. 10: Structure of a Firebase Analytics message. Each message consists of a sequence of bundles, each bundle contains the event information together with device and user information. In particular, each event is tagged with both the app Firebase ID, which is linked to the device Android ID when the app registers with Firebase Analytics, and the Google Advertising ID.

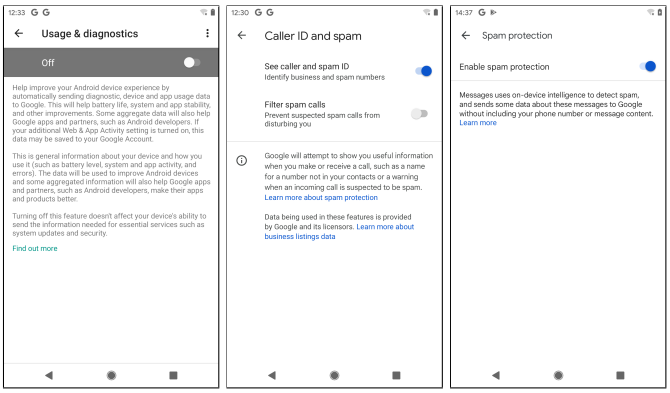
```
event {
  event_info {
    setting_code: "_o" // firebase_event_origin
    data_str: "app"
  }
  event_info {
    setting_code: "_sc" // firebase_screen_class
    data_str: "LegacyInCallActivity"
  }
  event_info {
    setting_code: "_si" // firebase_screen_id
    data_int: 6625442165989094212
  }
  event_code: "OUTGOING_CALL_PLACED"
  event_timestamp: 1635013242676
  previous_event_timestamp: 1635013037816
}
```

Fig. 11: Example of decoded Firebase Analytics event message when a phone call is made using the Google Dialer

The event itself is encoded as a protobuf and Figure 11 shows an example of a decoded event logged by the Google Dialer app when a phone call is made. The `event_code` value `OUTGOING_CALL_PLACED` records the fact that a call was made and the `event_timestamp` records the time when the call was made. The `event_info` value `LegacyInCallActivity` records the app screen (or "activity" in Android parlance) from which the call was made.

F. Reconstructed Protobuf Definitions and Decoding Software

The reconstructed protobuf definitions used here are available at https://github.com/doug-leith/android-protobuf-decoding, together with python scripts that can be used as a `mitmproxy` addon to parse recorded packet traces. For step by step instructions on how to collect decrypted packet traces from an Android handset see https://github.com/doug-leith/cydia.



(a) (b) (c)

Fig. 12

IV. Experimental Setup

A. Hardware and Software Used

Mobile handsets: Google Pixel 2 running Android 11 (build RP1A.201005.004.A1) with Google Play Services ver. 21.39.16 (150400-402663742) rooted using Magisk v23.0. Google Dialer ver. 70.05.401408800, Google Messages ver. 10.0.014 (Isengard_RC01.phone_dynamic). Although we only present measurements for Android 11 we also collected measurements from a Google Pixel 4a running Android 12 (build SP1A.210812.015), Google Play Services ver. 21.39.17 (190400-405802548), Google Dialer ver. 68.0.392726590, Google Messages ver 8.4.041. The behaviour observed is almost identical to that of Android 11.

WiFi access point: Raspberry Pi 4 Model B Rev 1.2/Raspbian GNU Linux 11/Mitmproxy 6.0.2 with iptables firewall configured to redirect HTTP/S traffic to port 8080 (on which mitmproxy listens) and also to block UDP traffic on HTTPS port 443 (so as to force any Google QUIC traffic to fall back to using TCP since we have no tools for decrypting QUIC).

B. Device Settings

At the start of each test we removed any SIM card and reflashed the handset with a fresh factory image. Following this, the handset reboots to a welcome screen and the user is then presented with a number of option screens. We note that all of the option toggle switches default to the opt-in choice, and so it is necessary for the user to actively select to opt-out. To mimic a privacy conscious user, we unchecked any of the options that asked to share data and only agreed to mandatory terms and conditions. Specifically, we deselected the (i) "Free up space" option, (ii) "Use location" option, (iii) "Allow scanning" option and (iv) the "Send usage and diagnostic data" option, see Figure 12(a). Note that there is no option to deselect automatic updates. We did not log in to Google user account during the onboarding process. After onboarding we inserted a SIM.

C. Test Design

Following previous mobile handset privacy studies [3], [4] we assume a privacy-conscious but busy/non-technical user,

who when asked, does not select options that share data but otherwise leaves handset settings at their default values. This provides a baseline for privacy analysis, and we expect that the level of data sharing may well be larger for a less privacy-conscious user.

Both the Google Dialer and Messages app include spam detection/protection services. By default these are enabled for both apps, but can be disabled by a user via the settings menu in each app. To explore the impact of these services on data sharing we take measurements both with spam detection/protection enabled (the default) and with it disabled. Google documentation[17] also suggests that these spam detection/protection services may treat calls/messages from phone numbers that are already in the handset contacts database differently from numbers that not in handset contacts database.

With these considerations in mind we carry out the following experiments:

Phone number in contacts:

1) Start a pair of handsets following a factory reset (mimicking a user receiving a new phone), insert a SIM in each handset and disable mobile data.

2) Login in to a Google account. This downloads a list of contacts, including the calling number used in our tests.

3) Make/receive phone calls and send/receive SMS messages between the pair of handsets. Record the network activity.

4) Disable the "Caller ID and spam detection" option in Google Dialer and the "Spam protection" option on Google Messages (both default to "on", see Figure 12)

5) Make/receive calls and send/receive SMS messages. Record the network activity.

Phone number not in contacts: As above, but do not login to Google account (the handset contacts database will be empty).

Interacting With Apps: During the above tests we interact with the apps to send/receive SMS messages and make/receive phone calls. Since our measurements in these tests established that user interactions (screens viewed, buttons clicked) are logged and sent to Google by the apps we then also additionally carried out tests where we (i) viewed the call history, (ii) viewed recent calls/favourites, (iii) viewed/edited contact details, (iv) opened the in-app settings menu and viewed the settings screens, (v) entered both text and phone numbers in the app search bar and

App Privacy Policy: We also tried to view the app privacy policy.

[17]"Your chats stay private with spam detection", Google Support page https://support.google.com/messages/answer/9327903.

V. Results: Google Messages

A. Inserting SIM

When a SIM is inserted into the handset Google Messages records this event via the Google Play Services ANDROID_MESSAGING log source:

```
event {
 eventType: BUGLE_TELEPHONY_EVENT
 bugleTelephonyEvent {
  carrierInfo {
    simStatus: LOADED
    simInfoNotUpdated: true
    simOperator: "27211"
    subscriptions {
      usingDefaultDataSubscriptionId: true
      numSimSlots: ONE
      DefaultSubscriptions {
        usingDefaultVoiceSubscriptionId: true
        usingDefaultSmsSubscriptionId: true
        usingDefaultDataSubscriptionId: true
      }
    }
    simSerialNumber: "89353111802...65506"
    simCarrierId: -1
}}}
```

and also via the Google Play Services CARRIER_SERVICES log source:

```
event {
  <...>
  packageVersionName: "10.0.014 (Isengard_RC01.phone_dynamic)"
  <...>
  simOperator: "27211"
  <...>
  simSerialNumber: "89353111802...65506"
  <...>
}
```

The `packageVersionName` value is the version name of Google Messages app. The `simOperator` value specifies the SIM operator (in this case 48 Mobile Ireland). The `simSerialNumber` value is the SIM card serial number or ICCID, which uniquely identifies the SIM card. Since these event records are also tagged with the handset AndroidID (see Figure 6) they act to link the handset and the SIM. Additionally, Google Play Services also separately sends SIM details and the AndroidID to https://android.clients.google.com/fdfe/uploadDynamicConfig, see [3], [4].

B. Sending/Receiving An SMS Message

We present measurements when sending an SMS message between two handsets using Google Messages with the spam protection service disabled and the handset phone numbers not in their contacts list. However, we note that in our tests similar behaviour was also observed when spam protection is enabled and/or the handset phone numbers are in the contacts list.

1) ANDROID_MESSAGING log source: On the handset sending a text we observe, for example, the following sequence of event messages sent by Google Messages via the Google Play Services ANDROID_MESSAGING log source[18]:

```
1635968886592 BUGLE_MESSAGE bugleMessageStatus: CREATED
1635968886593 BUGLE_APP_CONFIGURATION
1635968886600 BUGLE_COMPOSE
1635968886623 BUGLE_P2P_SUGGESTION suggestionEventType: SENT_MESSAGE
1635968886735 BUGLE_MESSAGE bugleMessageStatus: MESSAGE_ID_CREATED
1635968887029 BUGLE_P2P_SUGGESTION suggestionEventType: REQUEST
1635968887562 BUGLE_MESSAGE bugleMessageStatus: SENT sha256HashMsg: "
247836537599431109" sha256HashPrevMsg: "20042845847518237l"
```

The first BUGLE_MESSAGE event records the fact that a new message is created, the last BUGLE_MESSAGE event

[18]Each event message is similar to that in Figure 7 but for clarity and to save space we just show selected values from each message

records the fact that the message was successfully sent. The BUGLE_APP_CONFIGURATION event records the orientation of the screen and whether the handset is in multi-window mode. The other events log internal app processing steps as a new message is sent for transmission.

At the receiving handset we observe the following corresponding sequence of event messages:

```
1635968888138 BUGLE_P2P_SUGGESTION suggestionEventType: REQUEST
1635968888460 BUGLE_MESSAGE bugleMessageStatus: RECEIVED sha256HashMsg: "
2478365375994311O9" sha256HashPrevMsg: "200428458475182371"
1635968888685 BUGLE_P2P_SUGGESTION suggestionEventType: RECEIVED_MESSAGE
1635968890295 BUGLE_NOTIFICATION
1635968890295 BUGLE_APP appLaunch: VIA_NOTIFICATION
1635968890426 BUGLE_CONTACT_BANNER
1635968890712 BUGLE_MESSAGE bugleMessageStatus: READ
```

The first BUGLE_MESSAGE records the receipt of the message. The BUGLE_P2P_SUGGESTION events record processing of the message by the Google suggestions service (which can suggest links for more information related to a message, quick replies etc[19]). The BUGLE_APP event records launch of the app by the user clicking on the message arrival notification, and the final BUGLE_MESSAGE event records the fact that the received message has been displayed.

2) CARRIER_SERVICES log source: On the receiving handset the phone number of the SMS sender is transmitted to Google via the Google Play Services CARRIER_SERVICES log source, e.g.

```
timestamp: 1635968888300
event {
<...>
packageVersionName: "10.0.014 (Isengard_RC01.phone_dynamic)"
<...>
      incomingPhoneNumber: "+353872...351"
<...>
}
```

The `packageVersionName` value is the version name of Google Messages app. When a pair of handsets engage in a back-and-forth exchange of SMS messages, each handset sends the phone number of the other to Google via the CARRIER_SERVICES log source. Identifying a pair of communicating handsets (which is feasible, see below) therefore allows the phone numbers of both phones to be discovered.

3) Google Analytics Event Logging: On the handset sending a text an event message is sent to Google Analytics to record this, e.g.[20]:

```
1635968887562 data_str: "ConversationActivity" event_code: "ACTIVE_EVENT"
package_name: "com.google.android.apps.messaging"
google_ad_id: "916c714a-e838-479d-a7a6-3325d838da5f"
firebase_instance_id: "eVpvvohEDCqhfIGC7pXLnv"
```

On the receiving handset a corresponding event message is also sent to Google Analytics, e.g.

```
1635968894940 data_str: "ConversationActivity" event_code: "ACTIVE_EVENT"
package_name: "com.google.android.apps.messaging"
google_ad_id: "0fcb9970-3c60-426d-8186-452793942752"
firebase_instance_id: "fkT8O_dZhqxcNAfYucplGA"
```

These Google Analytics event messages act to link the SMS message exchange to the Google Advertising IDs of the handsets, and so to other advertising-related data held by Google and also potentially to third parties (recent measurements reveal that, for example, Samsung, Xiaomi, Realme and Microsoft all silently log handset data tagged with the Google Advertising ID via pre-installed system apps [4]).

[19]https://support.google.com/messages/answer/9265111?hl=en

[20]Each event message is similar to that in Figures 10 and 11 but to save space we just show selected values from each message

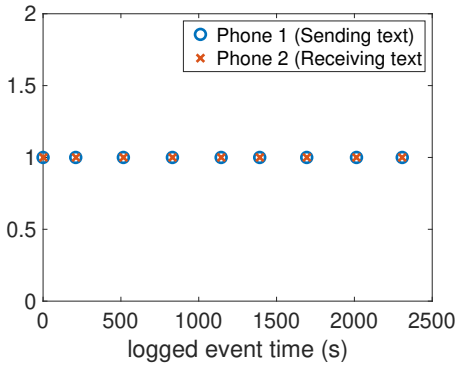


Fig. 13: Example of Google Messages log entry timestamps on a pair of communicating handsets. The x-axis is the logged event timestamp (rescaled from milliseconds to seconds and offset so the first entry has timestamp 0).

4) Using Timing To Identify Pairs Of Communicating Handsets: When the sender and receiver in a text conversation are both using Google Messages, then the time when the message was sent and the time when it was received are both sent to Google via the above event logging messages. This information can potentially be used to identify pairs of handsets engaged in an SMS text conversation. For example, Figure 13 shows the send and receive times sent to Google by a pair of handsets as a sequence of SMS messages are sent (at roughly 5 minute, or 300 second, intervals). On modern cellular networks the delay between sending and receiving a text is small (about 1 second in this experiment) and so both handsets log events with similar timestamps. Hence, given log event timestamp data such as that in Figure 13 it is possible to infer whether two handsets are communicating.

5) Using Message Hashes To Identify Pairs Of Communicating Handsets: Google Messages also sends to Google a signature of each message sent/received that uniquely identifies the message. Observe that the `sha256HashMsg` and `sha256HashPrevMsg` values logged by the sender and receiver in the above measurements are the same.

The `sha256HashMsg` value is derived from the SHA256 hash of the time, in hours since 1st Jan 1970, when the message was sent/received concatenated with the message content i.e the message text. This SHA256 hash is 32 bytes long, the lower 8 bytes are converted to a long int and then to a decimal string, which gives the `sha256HashMsg` value. Pseudo-code for the hash calculation is as follows:

```
byte[] bytes1 = String.valueOf(timestamp_ms / TimeUnit.HOURS.toMillis(1)).
getBytes("UTF-8")
byte[] bytes2 = messageText.getBytes("UTF-8")
byte[32] hash1 = SHA256(concat(bytes2, bytes1))
long hash2 = (long) (hash1[0] & 255)
for (i=1; i<8; i_++) {
  hash2 |= ((long) hash1[i] & 255) << (i*8)
}
return String.valueOf(hash2)
```

User Interaction	*ANDROID_MESSAGING Events Recorded*
Open app by (i) clicking icon, (ii) clicking message notification	BUGLE_APP VIA_LAUNCH_ICON, VIA_NOTIFICATION
Message create, read	BUGLE_MESSAGE CREATED, READ
View a conversation	BUGLE_CONVERSATION
Enter text in search box	BUGLE_SEARCH SEARCHBOX, FILTER_CLICKED, SEARCH_QUERY
View home screen	HOME_SCREEN
Contact: view, add, delete	BUGLE_CONTACTS_EVENT

TABLE II: ANDROID_MESSAGING events observed recording user interactions with Google Messages app. When an event is recorded, the time, event name and relevant app/message information (e.g. conversation ID) are sent to Google. Events are tagged with the handset AndroidID.

The `sha256HashPrevMsg` value is the hash for the previous message sent/received in the conversation.

These `sha256HashMsg` and `sha256HashPrevMsg` values therefore act to uniquely identify the SMS messages sent. Identifying whether a pair of handsets using Google Messages are communicating therefore simply involves comparing the `sha256HashMsg` values sent by both handsets to Google.

We note that these hash values are not suitable for spam prevention since they contain the send/receive timestamp and so spam messages received at different times will have different hash values even though the message text is the same. Rather they seem crafted in way that facilitates linking both ends of a message exchange. We note also that the code where the hash values are calculated contains the descriptive text "E2EChatIntegrityMetricsHelperImpl". "E2E" is a commonly used technical abbreviation for "end-to-end" which is also suggestive that the purpose of the hashes is the linking of the two ends of a message exchange.

C. Interacting With Messages App

When a user interacts with the Google Message app, their actions are recorded and sent to Google both via the Google Play Services ANDROID_MESSAGING log source and via Google Analytics. Table II lists the events recording user interactions via ANDROID_MESSAGING that we observed in our measurements. The event names, which are extracted from the app itself and so are Google's own, are largely self-explanatory and include opening of the app, composing and reading a message, viewing a conversation (message exchanges between the same pair of handsets), entering text in the app search bar (where phone numbers, contact names etc are entered), navigating to the app home screen. Table III lists the events recording user interactions that we observed sent via Google Analytics. Events generally record the activity name (an activity being a screen within the app e.g. the HomeActivity is the main screen in the app), the time of the event and the duration.

We note that entering text (a phone number, contact name etc) in the app search bar is observed to generate logging of a cascade of internal Google Play Services events, including via the ICING and AUTOFILL_WITH_GOOGLE Google Play Services log sources. However, we leave decoding of this data sent to Google to future work.

User Interaction	Google Analytics Event
View an app screen	HomeActivity, ConversationActivity, ZeroStateSearchActivity, PeopleAndOptionsActivity, ApplicationSettingsActivity, SpamSettingsActivity, SmartsSettingsActivity, FederatedLearningSettingsActivity, AboutPrivacyTermsActivity
Send/receive a message	ACTIVE_EVENT

TABLE III: Google Analytics events observed recording user interactions with Google Messages app. When an event is recorded, the time, event name and duration are sent to Google. Events are tagged with the Google Advertising ID and app FirebaseID (which is linked to the handset AndroidID).

D. Viewing App Privacy Policy

As already noted, viewing the privacy policy of the Google Messages app is not straightforward. It is necessary to: (i) click on the three dots in search bar to open the Settings menu, (ii) scroll down to see an "About, terms and privacy" link (see Figure 3(a)), (iii) click on this to open a new menu that shows a "Privacy Policy" link, (iv) click on this link which opens a Google Chrome window. At this point, to proceed it is necessary to agree to the Google Chrome terms and conditions, see Figure 3(b). It is not possible to proceed to view the Messages app privacy policy without first agreeing to the additional Google Chrome terms and conditions.

At this point the Messages app silently sends messages to Google Analytics https://app-measurement.com/a logging the fact that the page with the privacy policy link has been viewed, e.g.

```
event_info {
  setting_code: "_pc" // firebase_previous_class
  data_str: "AboutPrivacyTermsActivity"
}
event_code: "_vs" // screen_view
event_timestamp: 1636311111608
}
package_name: "com.google.android.apps.messaging"
google_ad_id: "916c714a-e838-479d-a7a6-3325d838da5f"
firebase_instance_id: "eVpvvohEDCqhfIGC7pXLnv"
```

Agreeing to the Google Chrome terms and conditions loads the page at http://www.google.com/intl/en_IE/policies/privacy/ which redirects to https://policies.google.com/privacy?hl=en&gl=IE. During the loading of this page (i) 20 connections are made to www.youtube-nocookie.com/youtubei/v1/log_event sending what appears to be telemetry, (ii) a connection is made to download https://www.google-analytics.com/analytics.js, (iii) and then connections are made to www.google-analytics.com/j/collect, stats.g.doubleclick.net/j/collect and https://play.google.com/log:

```
POST https://www.google-analytics.com/j/collect?v=1&_v=j93&a=1137820265&t=
pageview&_s=1&dl=https://policies.google.com/privacy&dr=http://www.google.com
/&ul=en-ie&de=UTF-8&dt=Privacy Policy - Privacy & Terms - Google&sd=24-bit&sr
=412x732&vp=412x604&je=0&_u=YEBAAEABAAAAAC &jid=2016745238&gjid=857776913&cid
=1233068479.1636311067&tid=UA-28138501-1&_gid=1044709367.1636311067&_r=1&_slc
=1&z=1504113415
Headers
   referer: https://policies.google.com/

POST https://stats.g.doubleclick.net/j/collect?t=dc&aip=1&_r=3&v=1&_v=j93&tid=
UA-28138501-1&cid=1233068479.1636311067&jid=2016745238&gjid=857776913&_gid
=1044709367.1636311067&_u=YEBAAEAAAAAAC &z=1415398462
Headers
   x-client-data: COznygE=
   referer: https://policies.google.com/

POST https://play.google.com/log?format=json&hasfast=true
Headers
   x-client-data: COznygE=
   referer: https://policies.google.com/
 <post body appears to be telemetry>
```

VI. Results: Google Dialer

A. Making/Receiving A Phone Call

We now present measurements when making a phone calls between two handsets using Google Dialer with the Caller and Spam ID option disabled. When this option is enabled additional event messages are sent to Google, but we will describe these later.

1) ANDROID_MESSAGING log source: On the handset initiating the phone call we observe, for example, the following sequence of event messages sent by the Google Dialer via the Google Play Services ANDROID_DIALER log source[21]:

```
1635969033382 MAIN_CLICK_FAB_TO_OPEN_DIALPAD
1635969034630 searchQuery
1635969039257 queryLength: 1
1635969039478 queryLength: 2
1635969039881 queryLength: 3
1635969041305 queryLength: 4
1635969041680 queryLength: 5
1635969042060 queryLength: 6
1635969044085 queryLength: 7
1635969044556 queryLength: 8
1635969044906 queryLength: 9
1635969045359 queryLength: 10
1635969064139 PRECALL_INITIATED
1635969065267 TIDEPODS_STATUS_BAR_NOTIFICATION_SHOWED
1635969065297 TIDEPODS_BUBBLE_SHOWED
1635969085622 SCOOBY_CALL_LOG_SPAM_DISABLED
1635969085622 USER_PARTICIPATED_IN_A_CALL callDuration: 12344
1635969085720 ANNOTATED_CALL_LOG_FORCE_REFRESH_CHANGES_NEEDED
1635969085868 ANNOTATED_CALL_LOG_FORCE_REFRESH_NO_CHANGES_NEEDED
 1635969085918 ANNOTATED_CALL_LOG_NOT_DIRTY
```

To make the call the dialpad in the app is opened and the phone number typed. The MAIN_CLICK_FAB_TO_OPEN_DIALPAD event records opening of the dialpad, the next sequence of SearchQuery event messages record each individual keypess as the phone number is typed, and also the timing of these keypresses. The PRECALL_INITIATED event through to the TIDEPODS_BUBBLE_SHOWED record the internal process of initating the call over the phone network and displaying the in-call user interface. The USER_PARTICIPATED_IN_A_CALL event records the termination of the call and, amongst other things, sends the call duration to Google (the value is in milliseconds so a value of 12344 corresponds to a call of 12.344 seconds duration). The last three CALL_LOG events record internal actions associated with updating the handset call log.

At the receiving handset we observe the following corresponding sequence of event messages:

[21]Each event message is similar to that in Figures 8 and 9 but to save space we just show selected values from each message

```
1635969070066 CALL_SCREENING_SERVICE_MUSIC_IS_NOT_ACTIVE
1635969070096 INCOMING_CALL_SCREENED
1635969070639 TIDEPODS_BUBBLE_SHOWED
1635969070644 TIDEPODS_STATUS_BAR_NOTIFICATION_SHOWED
1635969072226 TIDEPODS_STATUS_BAR_NOTIFICATION_ANSWER
1635969085350 SCOOBY_CALL_LOG_SPAM_DISABLED
1635969085350 USER_PARTICIPATED_IN_A_CALL callDuration: 12865
1635969085483 ANNOTATED_CALL_LOG_FORCE_REFRESH_CHANGES_NEEDED
```

The first events record call screening, displaying a notification to the user that here is an incoming call and the user pressing the answer button. The USER_PARTICIPATED_IN_A_CALL event records the termination of the call and sends the call duration to Google. Note the close match in the call durations recorded by the sender and receiver i.e. 12.344 seconds and 12.865 seconds respectively. Presumably the small difference of 0.52 seconds is due to the telephone network delay between one phone hanging up and the other phone being informed of this.

2) Google Analytics Event Logging: On the handset initiating the phone call an event message is sent to Google Analytics to record this, e.g.

```
data_str: "LegacyInCallActivity" event_code: "OUTGOING_CALL_PLACED"
package_name: "com.google.android.dialer"
google_ad_id: "916c714a-e838-479d-a7a6-3325d838da5f"
firebase_instance_id: "f86VDMH_SSGcArMl6Up973"
```

At the receiving handset the incoming call is also logged to Google Analytics:

```
data_str: "LegacyInCallActivity" event_code: "INCOMING_CALL_RECEIVED"
package_name: "com.google.android.dialer"
google_ad_id: "0fcb9970-3c60-426d-8186-452793942752"
firebase_instance_id: "cyoEhnfBQtChaUDlrYRfYB"
```

These Google Analytics event messages act to link the phone call to the Google Advertising ID of the sender handsets, and so to other advertising-related data held by Google and also potentially to third parties.

3) Identifying Pairs Of Communicating Handsets: When the caller and receiver in a phone conversation are both using the Google Dialer, then the time when the time the call ended and the call duration are both sent to Google via the above event logging messages. This information can potentially be used to identify pairs of handsets engaged in phone conversations. For example, Figure 14 shows the call times and durations sent to Google by a pair of handsets as they engage in a sequence of phone calls (at roughly 5 minute, or 300 second, intervals). Clearly, by comparing the pattern of call times and call durations on a pair of handsets it is possible to infer whether two handsets are communicating.

4) Caller and Spam ID Enabled: When the Caller and Spam ID option is enabled we observe additional events sent to Google via the Google Play Services SCOOBY_EVENTS log source (Scooby in the internal name for the spam scanning service). For example:

```
timestamp: 1635013551317
event {
 1 {
   packageName: "Dialer"
   packageVersionName: "70.05.401408800"
   incomingPhoneNumber: "+353872...351"
   <...>
}
```

The `packageVersionName` value is the version name of Google Dialer app. Note that this SCOOBY_EVENTS message is sent every time a call is received and even if

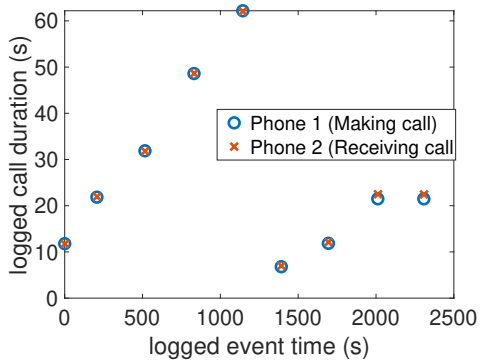


Fig. 14: Example of Google Dialer log entries on a pair of communicating handsets. The x-axis is the logged event timestamp (rescaled from milliseconds to seconds and offset so the first entry has timestamp 0), the y-axis is the logged callDuration value (again rescaled from milliseconds to seconds).

the phone number is in the handset contacts database. When a pair of handsets engage in a back-and-forth phone calls and both have the Caller and Spam ID option enabled, then each handset sends the phone number of the other to Google via the SCOOBY_EVENTS log source. Identifying a pair of communicating handsets (which seems feasible based on the data we observe to be collected by Google, see above) therefore allows the phone numbers of both phones to be discovered.

B. Interacting With Dialer App

Similarly to Google Messages, when a user interacts with the Google Dialer app, their actions are recorded and sent to Google both via the Google Play Services ANDROID_DIALER log source and via Google Analytics. Table IV lists the events recording user interactions via ANDROID_DIALER that we observed in our measurements. App screens viewed are also logged via Google Analytics.

C. Viewing App Privacy Policy

The Google Dialer does not appear to have any app privacy policy link. The only privacy policy link that we could find was by following these steps: (i) click on the three dots in search bar to open ta menu, (ii) click on the ”Help and feedback” to open the Support page, (iii) click on the three dots at the top right of the Support page to open a new menu, (iv) click on the ‘Privacy Policy” link that is revealed but this privacy policy appears to be associated with the Support page rather than the Dialer app. As with the Google Messages app this process opens a Google Chrome window and it is necessary to agree to the additional Google Chrome terms and conditions in order to proceed. The user eventually arrives at the web page http://www.google.com//policies/privacy/ which redirects to https://policies.google.com/privacy. This is Google’s global privacy policy page, with no app specific information and not localised to Europe/Ireland (unlike for the Messages app).

User Interaction	*ANDROID_MESSAGING Events Recorded*
Start/finish a phone call	PRECALL_INITIATED, USER_DID_NOT_PARTICIPATE_IN_CALL, USER_PARTICIPATED_IN_CALL
Answer call	TIDEPODS_STATUS_BAR_NOTIFICATION_ANSWER
Open dialpad	MAIN_CLICK_FAB_TO_OPEN_DIALPAD
Hang up	IN_CALL_DIALPAD_HANG_UP_BUTTON_PRESSED
Search for contact, type a phone number	MAIN_CLICK_SEARCH_BAR, searchQuery
View contacts	MAIN_SWITCH_TAB_TO_CONTACTS
View favourites	MAIN_SWITCH_TAB_TO_FAVORITE, MAIN_OPEN_WITH_TAB_FAVORITE
View call log	MAIN_SWITCH_TAB_TO_CALL_LOG, CALL_LOG_LAUNCHED, CALL_LOG_SHOW_POPUP_MENU, closeCallLog
Open menu	MAIN_TOOLBAR_SHOW_MENU
View call history	MAIN_TOOLBAR_MENU_OPEN_CALL_HISTORY

TABLE IV: ANDROID_DIALER observed events recording user interactions with Google Dialer app. When an event is recorded, the time, event name and relevant app/message information are sent to Google. Events are tagged with the handset AndroidID.

During this process a number of requests are made to www.google.com/tools/feedback/mobile, presumably associated with pre-loading Support page data. Some of the responses set a cookie but these appear to be scrubbed since they are not resent in later requests. Similarly to the Google Messages app, loading the privacy policy page prompts multiple connections to www.youtube-nocookie.com/youtubei/v1/log_event sending what appears to be telemetry, and to www.google-analytics.com/j/collect, stats.g.doubleclick.net/j/collect.

VII. Lack Of Anonymity

The data logging that we report here is, for many people, not anonymous since it can be directly linked to their online and real-world identities. This can happen in several ways:

A. Android ID

All of the events recorded via the Google Play Services Clearcut logger are tagged with the handset’s Android ID.

Via the data that is regularly sent to https://android.googleapis.com/checkin by Google Play Services the handset Android ID is linked to (i) the handset hardware serial number, (ii) the SIM IMEI (which uniquely identifies the SIM slot) and (iii) the user’s Google account, see Section III-D. Via the data sent when a SIM is inserted the Android ID is also linked to the SIM serial number/ICCID, which uniquely identifies the SIM card, see Section V-A. By making a request using https://takeout.google.com/ for the data associated with the Google user account used in our tests we further confirmed that the device Android ID is linked with the user Google account and device/SIM identifiers. Namely, the data reported under the heading “Android Device Configuration Service” includes the

Android ID for each handset used as well as the handset serial number, SIM IMEI, last IP address used and mobile operator details.

When creating a Google account it is necessary to supply a phone number on which a verification text can be received. For many people this will be their own phone number[22], thereby linking their Google account to their phone number. Use of Google services such as buying a paid app on the Google Play store or using Google Pay further links a person's Google account to their credit card/bank details. A user's Google account, and so the Android ID, can therefore commonly be expected to be linked to the person's real identity.

Additionally, when a message is received by the Google Messages app the sender's phone number is sent to Google via the Google Play Services Clearcut logger, see Section V-B2. By combining data from the pair of handsets involved in an exchange of messages (which seems perfectly feasible based on the hashes of the message text that we observe to be collected) both phone numbers may be revealed and linked to the Android IDs. Similarly when the spam protection option is enabled in the Google Dialer (as it is by default), see Section VI-A4.

The value of the Android ID can be changed, but this requires carrying out a full factory reset of the handset. Further, unless the handset, SIM card and Google account are also changed at the same time then the new Android ID value can be easily relinked to the handset/SIM/user via the data sent to Google by Google Play Services, as already noted.

B. Google Advertising ID

All of the events recorded via Google Analytics are tagged with the user's Google Advertising ID and the sender app's Firebase ID. The app Firebase ID is directly linked to the handset Android ID when the app registers to use the Google Analytics service, see Section III-E1, and so to the handset/SIM/user. The Google Advertising ID is used to link together other data collected by Google for advertising purposes and and also potentially by third parties (recent measurements reveal that, for example, Samsung, Xiaomi, Realme and Microsoft all silently log handset data tagged with the Google Advertising ID via pre-installed system apps [4]). Depending on the advertising-related data collected (e.g. if payment/purchase or account data is collected), this can lead to the Google Advertising ID becoming de-anonymised.

The Google Advertising ID can be reset by the handset user, but this needs to be done manually and so is probably an infrequent occurence.

C. Handset Phone Number

When a message is received by the Google Messages app the sender's phone number is sent to Google, see Section V-B2. By combining data from the pair of handsets involved in an exchange of messages both phone numbers are therefore revealed. Similarly, when the spam protection option is enabled in the Google Dialer (as it is by default), see Section VI-A4

[22]While it is possible to use online services that use a shared phone number to temporarily receive texts, or to buy a scratch SIM that is later discarded, in practice it seems unlikely that many people would take such steps.

VIII. Summary

We report on measurements of the data sent to Google by the Google Messages and Google Dialer apps on an Android handset. We find that these apps tell Google when message/phone calls are made/received. The data sent by Google Messages includes a hash of the message text, allowing linking of sender and receiver in a message exchange, and by Google Dialer the call time and duration, again allowing linking of the two handsets engaged in a phone call. Phone numbers are also sent to Google. In addition, the timing and duration of user interactions with the apps are sent to Google. There is no opt out from this data collection. The data is sent via two channels, the Google Play Services (i) Clearcut logger and (ii) Google/Firebase Analytics. This study is therefore one of the first to cast light on the actual telemetry data sent by Google Play Services, which to date has largely been opaque.

References

[1] "Google Safe Browsing API (v4)," 2020. [Online]. Available: https://developers.google.com/safe-browsing/v4

[2] D. J. Leith, "Web Browser Privacy: What Do Browsers Say When They Phone Home?" *IEEE Access*, 2021.

[3] ——, "Mobile Handset Privacy: Measuring The Data iOS and Android Send to Apple And Google," in *Proc Securecomm*, 2021.

[4] H. Liu, P. Patras, and D. J. Leith, "Android Mobile OS Snooping By Samsung,Xiaomi, Huawei and Realme Handsets," in *SCSS Tech Report, Oct 2021*, 2021. [Online]. Available: https://www.scss.tcd.ie/doug.leith/Android_privacy_report.pdf

[5] D. J. Leith and S. Farrell, "Contact Tracing App Privacy: What Data Is Shared By Europe's GAEN Contact Tracing Apps," in *Proc IEEE INFOCOM*, 2021.

[6] A. Cortesi, M. Hils, T. Kriechbaumer, and contributors, "mitmproxy: A free and open source interactive HTTPS proxy (v5.01)," 2020. [Online]. Available: https://mitmproxy.org/

[7] "Learn about the Android Device Configuration Service, Google Help Pages," Accessed 5 August 2020. [Online]. Available: https://support.google.com/android/answer/9021432?hl=en