What Data Do the Google Dialer and Messages Apps on Android Send to Google?

Douglas J. Leith(✉)

Trinity College Dublin, Dublin, Ireland
doug.leith@tcd.ie

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 work Google say that they plan to make multiple changes to their Messages and Dialer apps.

1 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.

[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.

This work was supported by Science Foundation Ireland grant 16/IA/4610.

F. Li et al. (Eds.): SecureComm 2022, LNICST 462, pp. 549–568, 2023.
https://doi.org/10.1007/978-3-031-25538-0_29

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.

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 handsets [7, 9]. 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 [7,9].

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. 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).

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

1.1 GDPR

We report on a technical study here, not a legal one. 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 and so it is European data regulations that apply). Roughly speaking, there are three main basis under GDPR for data collection[3]: (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.

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 Sect. 5.2. 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 Sect. 6.1.

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.

The linkage between the various identifiers is illustrated schematically on Fig. 1.

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

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.

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[4]. 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.

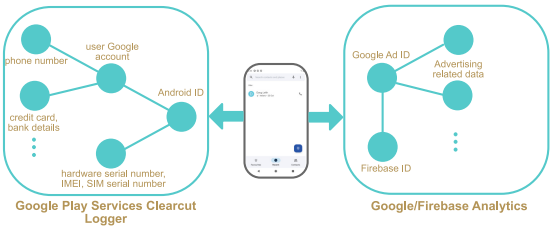


Fig. 1. Illustrating how handset data can be linked to a person's real identity. Handset data sent to Google via the Google Play Services Clearcut logger is tagged with the Android ID, which in turn is linked to the user's Google account and to device/SIM identifiers. The user's Google account in turn 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.

1.2 Lack of App-Specific Privacy Policy

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, (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 to view the privacy policy web page at http://www.google.com/intl/en_IE/policies/privacy/ (note

[4] 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/.

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 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 Sect. 5.4.

Google Dialer. The Google Dialer does not appear to have an app privacy policy link, only a privacy policy associated with the Support pages.

1.3 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/received message 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.

2 Related Work

While the Android ecosystem continues to evolve, most smartphone users remain largely unaware of the personal information disclosed by their devices and the apps they run [13]. This has motivated extensive privacy and security over recent years, e.g. see [4–6,10–12,14] and references therein, and triggered data protection legislation with nearly 100 articles laying out privacy requirements [3]. Its worth noting that much of this previous work uses static analysis i.e. inspection of the app binary to infer permissions used etc. While this scales well, allowing inspection of many apps, it essentially highlights potential rather than actual app behaviour i.e. can lead to "false positives" re privacy leakage. In the present work we employ dynamic analysis i.e. inspect the output of the running app. This has the great advantage that since actual app behaviour is recorded there are privacy false positives and stronger conclusions can therefore be drawn.

Probably closest to the present work are recent analyses of the data shared by Google Play Services [7–9]. The measurement study in [8] 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 [7] extended consideration to the data sent to Apple by an iPhone/IOS. Recently, in [9] 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 [7,8] 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× that uploaded by the mobile OS developer, rising to around 30× 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[5] 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

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

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.

3 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

3.1 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 [2] and adjust the 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.

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.

3.2 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 Fig. 2.

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.

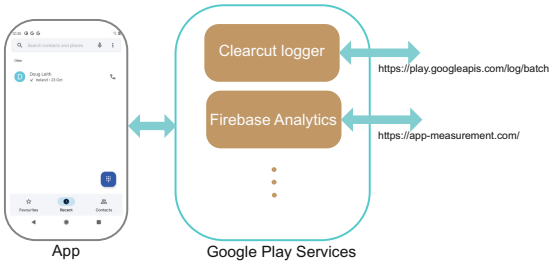


Fig. 2. Schematic illustrating app data flow. The app sends event data to Google Play Services via the Clearcut logger and Firebase Analytics APIs. These Google Play Services components then batch up the data and send it to Google servers. Note that Google Play Services provides many other APIs and services in addition to the Clearcut logger and Firebase Analytics.

3.3 Decoding Google 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. The message body is encoded in a binary protobuf format[6]. Figure 3 shows the structure of the decoded message, including an example header message. 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.

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

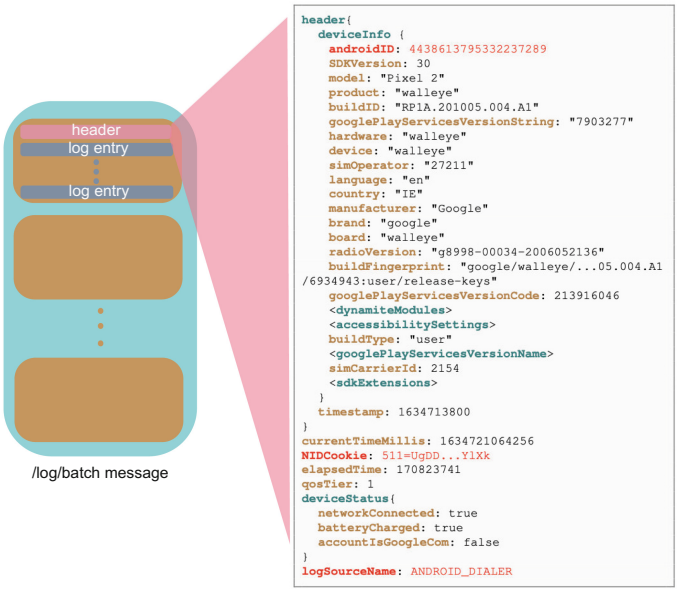


Fig. 3. Structure of messages sent to play.googleapis.com/log/batch by the Google Play Services Clearcut logger. Each message consists of one or more bundles of log entries, indicated in brown. Each bundle has a header containing device details and persistent identifiers (Google androidID, NID cookie) 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.

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 4(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 Messages app contains more than 2000 distinct protobufs[7].) and

[7] 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.

then (iii) trace back within the code to determine how the value of each entry in the protobuf is calculated. Figure 4(b) shows the result of this fairly laborious process.

```
logEntry:{
1: 1635013045967
6 {
 1: 2
 3 {
  1: 1
  2: 1
  <...>
  5 {
   1: 1635013045028
   2: 898
  }
  8: 2
  9: 1282237833693804524
  12: 1
  14: 2
  16: 2
  17: 4
  18: 10
  19: 4
  20 {
   1: 30524580
   3: ""
  }
  21: 778
  27: 8480061162880308485
  31: "NOT_AVAILABLE"
  33: "-8443536869600326524"
  34: "5478611868067030819"
  <...>
```

(a)

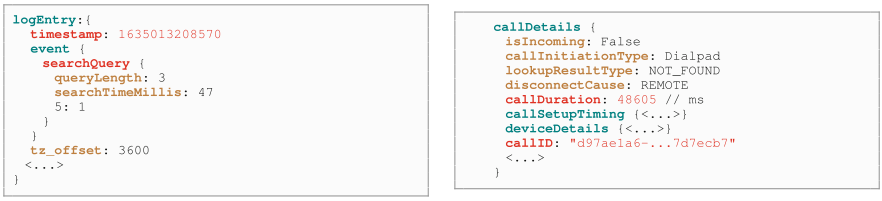
```
logEntry:{
timestamp: 1635013045967
event {
 eventType: BUGLE_MESSAGE
 bugleMessage {
  messageProtocol: ONE_ON_ONE
  bugleMessageStatus: SENT
  <...>
  messageTiming {
   currentTime_ms: 1635013045028
   elapsedTimeSinceMsgSendRecv_ms: 898
  }
  bugleMessageSource: CONVERSATION_ACTIVITY
  usageStatsLoggingId: 1282237833693804524 8
  conversationType: ONE_ON_ONE
  sendAttempt: FIRST_ATTEMPT_TO_SEND
  wasRCS: HAS_ALWAYS_BEEN_XMS_CONVERSATION
  rcsStatus: RCS_AVAILABILITIES_ISSUES
  configStatus: CARRIER_SETUP_PENDING
  phoneNumberFormat2: PHONENUMBER
  carrierServicesData {
   versionCode: 30524580
   3: ""
  }
  messageSendClickToSentLatency: 778
  conversationIdSHA1: 8480061162880308485
  RcsConfig: "NOT_AVAILABLE"
  sha256HashMsg: "-8443536869600326524"
  sha256HashPrevMsg: "5478611868067030819"
 <...>
```

(b)

Fig. 4. 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.

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



(a) (b)

Fig. 6. 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).

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 5 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 Fig. 5(a) for an example log entry sent in response to a keypress. When a phone call finishes this event is also logged, using a message similar to that in Fig. 5 but with `AOSPEventType` value `USER_PARTICIPATED_IN_A_CALL` and including additional data recording call details, including the call duration (in milliseconds), see Fig. 5(b).

3.4 Decoding Google/Checkin Message

Google Play Services sends periodic messages to https://android.googleapis.com/checkin that act to link together a number of persistent device and user identifiers, see [1,7–9]. 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.

4 Experimental Setup

4.1 Hardware and Software Used

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. 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.

4.2 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. 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. 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.

4.3 Test Design

Following previous mobile handset privacy studies [7,9] 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[8] 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.

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

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

5 Results: Google Messages

5.1 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"
    <...>
    simSerialNumber: "89353111802...65506"
    simCarrierId: -1
}}}
```

and similarly via the Google Play Services CARRIER_SERVICES log source. 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 Fig. 3) 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 [7,9].

5.2 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.

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[9]:

```
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: "200428458475182371"
```

The first BUGLE_MESSAGE event records the fact that a new message is created, the last BUGLE_MESSAGE event 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: "247836537599431109" 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.[10]). 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.

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"
<...>
}
```

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.

[9] Each event message is similar to that in Fig. 4 but for clarity and to save space we just show selected values from each message.

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

Google Analytics Event Logging. On the handset sending a text an event message is sent to Google Analytics to record this, e.g.[11]:

```
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.

Using Hashes to Identify Communicating Phones. 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. The `sha256HashPrevMsg` value is the hash for the previous message sent/received in the conversation.

These `sha256HashMsg` and `sha256HashPrevMsg` values can therefore act to identify the SMS messages sent. Identifying whether a pair of handsets using Google Messages are communicating therefore simply involves comparing the pair of `sha256Hash` values sent by both handsets to Google. Even if there occasional pairs of hash collisions (the same pair of short text messages is sent around the same time by two handsets), it only needs one hash miss to reveal that a pair of handsets are communicating.

5.3 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. The events logged 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.

[11] To save space we just show selected values from each message.

5.4 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, (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 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, https://stats.g.doubleclick.net/j/collect and https://play.google.com/log.

6 Results: Google Dialer

6.1 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.

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[12]:

[12] Each event message is similar to that in Figs. 5 and 6 but to save space we just show selected values from each message.

```
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 s 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:

```
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 s and 12.865 s respectively. Presumably the small difference of 0.52 s is due to the telephone network delay between one phone hanging up and the other phone being informed of this.

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_SSGcArM16Up973"
```

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.

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, Fig. 7 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 min, or 300 s, intervals). Clearly, by comparing the pattern of call times and call durations on a pair of handsets it may be possible to infer whether two handsets are communicating.

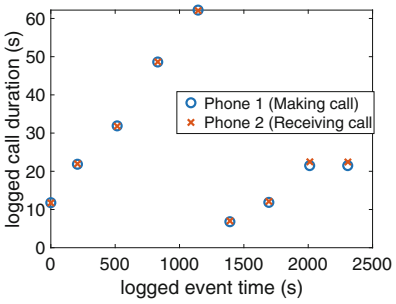


Fig. 7. Example of Google Dialer log entries on a pair of communicating handsets. The x-axis is the logged event timestamp (rescaled from milliseconds to seconds and offset so the first entry has timestamp 0), the y-axis is the logged callDuration value (again rescaled from milliseconds to seconds).

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 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.

6.2 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.

7 Summary

We report on measurements of the data sent to Google by the Google Messages and Google Dialer apps. 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 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. Learn about the Android Device Configuration Service, Google Help Pages (Accessed 5 Aug 2020). https://support.google.com/android/answer/9021432?hl=en
2. Cortesi, A., Hils, M., Kriechbaumer, T., contributors: mitmproxy: A free and open source interactive HTTPS proxy (v5.01) (2020). https://mitmproxy.org/
3. European Parliament and Council of the European Union: Regulation on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (data protection directive) (2016)
4. Gamba, J., Rashed, M., Razaghpanah, A., Tapiador, J., Vallina-Rodriguez, N.: An analysis of pre-installed android software. In: IEEE Symposium on Security and Privacy (S&P), pp. 1039–1055 (2020)
5. Jia, Q., Zhou, L., Li, H., Yang, R., Du, S., Zhu, H.: Who leaks my privacy: Towards automatic and association detection with gdpr compliance. In: Biagioni, E.S., Zheng, Y., Cheng, S. (eds.) Wireless Algorithms, Systems, and Applications, pp. 137–148 (2019)

6. Jin, H., et al.: Why are they collecting my data? inferring the purposes of network traffic in mobile apps. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. **2**(4), 1–27 (2018)
7. Leith, D.J.: Mobile handset privacy: measuring the data ios and android send to apple and google. In: Garcia-Alfaro, J., Li, S., Poovendran, R., Debar, H., Yung, M. (eds.) SecureComm 2021. LNICST, vol. 399, pp. 231–251. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90022-9_12
8. Leith, D.J., Farrell, S.: Contact Tracing App Privacy: What Data Is Shared By Europe's GAEN Contact Tracing Apps. In: Proc IEEE INFOCOM (2021)
9. Liu, H., Patras, P., Leith, D.J.: Android Mobile OS Snooping By Samsung,Xiaomi, Huawei and Realme Handsets. SCSS Tech Report, Oct 2021 (2021). https://www.scss.tcd.ie/doug.leith/Android_privacy_report.pdf
10. Razaghpanah, A., Nithyanand, R., Vallina-Rodriguez, N., Sundaresan, S.: Apps, Trackers, Privacy, and Regulators: A Global Study of the Mobile Tracking Ecosystem. In: Network and Distributed System Security Symposium (NDSS) (2018)
11. Reardon, J., Feal, Á., Wijesekera, P., On, A.E.B., Vallina-Rodriguez, N., Egelman, S.: 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In: 28th USENIX Security Symposium (USENIX Security 19), pp. 603–620. USENIX Association, Santa Clara, CA (Aug 2019). https://www.usenix.org/conference/usenixsecurity19/presentation/reardon
12. Ren, J., Lindorfer, M., Dubois, D.J., Rao, A., Choffnes, D., Vallina-Rodriguez, N.: Bug fixes, improvements,... and privacy leaks. In: Network and Distributed System Security Symposium (NDSS) (2018)
13. Van Kleek, M., Liccardi, I., Binns, R., Zhao, J., Weitzner, D.J., Shadbolt, N.: Better the devil you know: Exposing the data sharing practices of smartphone apps. In: CHI Conference on Human Factors in Computing Systems, pp. 5208–5220 (2017)
14. Zhang, D., Guo, Y., Guo, D., Wang, R., Yu, G.: Contextual approach for identifying malicious inter-component privacy leaks in android apps. In: IEEE Symposium on Computers and Communications (ISCC), pp. 228–235 (2017)