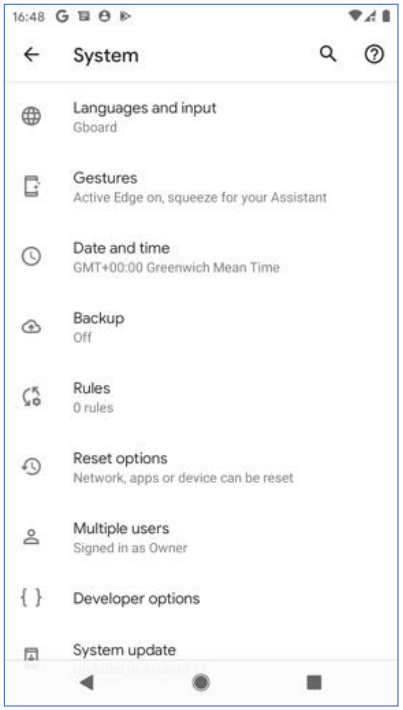
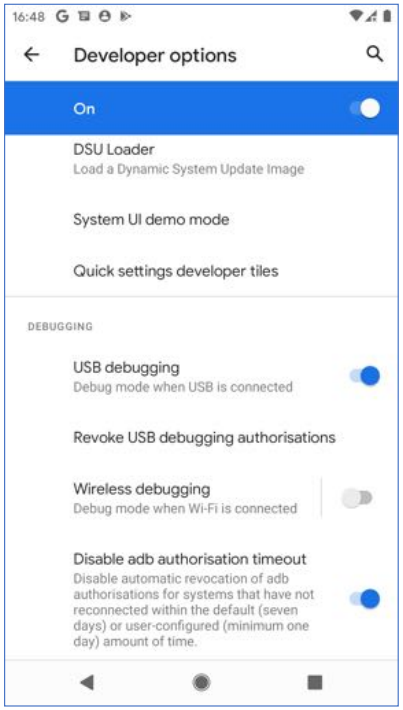
Looking inside your phone …





- Google Pixel 2 phone
- Enable developer options, USB debugging
- Connect to laptop by USB cable

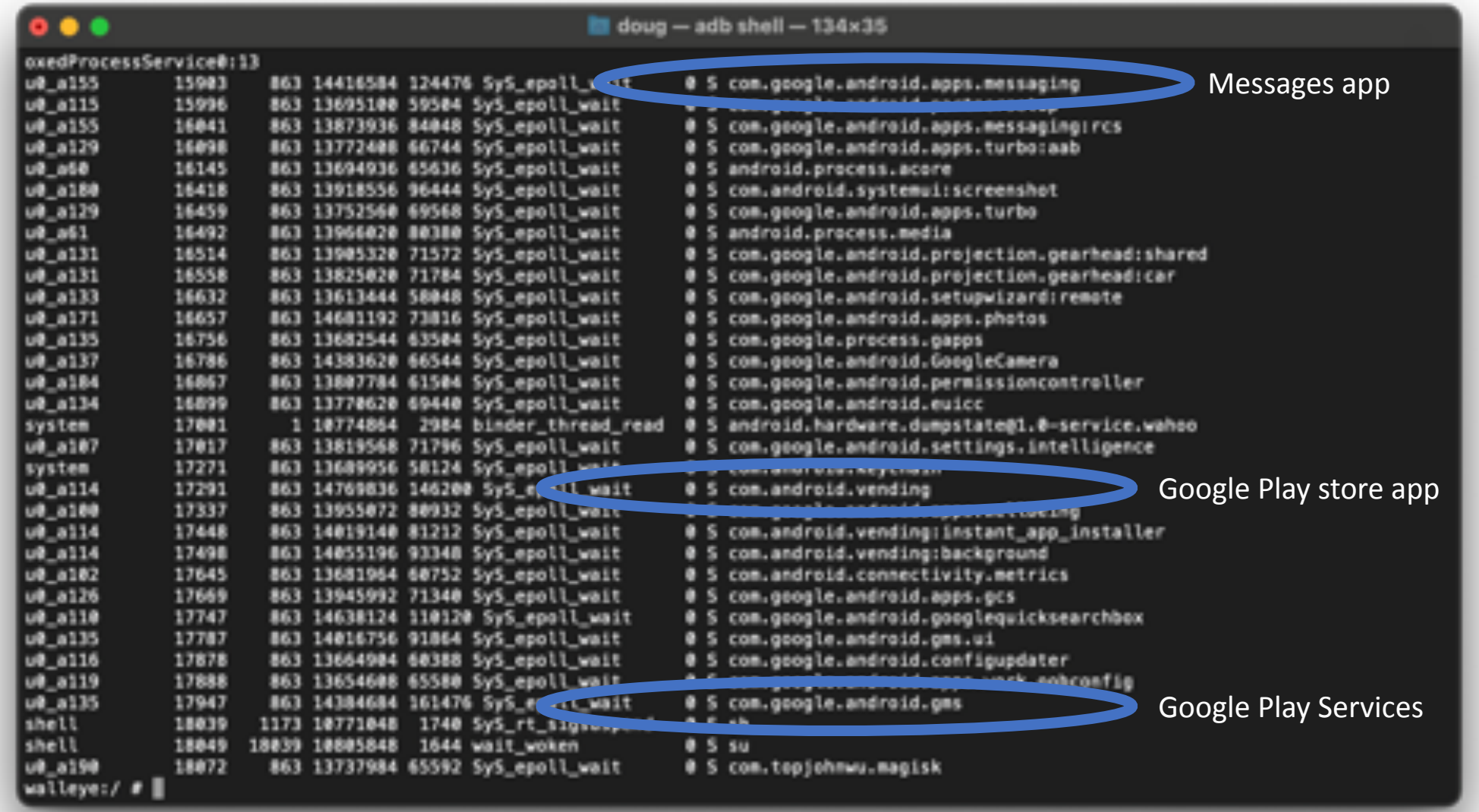
Looking inside your phone …

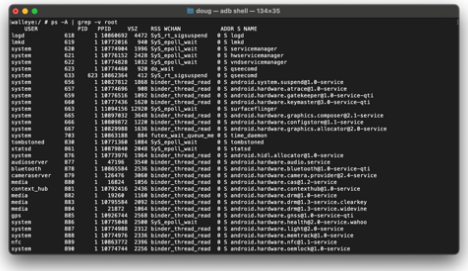
- Google Pixel 2 phone
- Enable developer options, USB debugging
- Connect to laptop by USB cable
- “adb shell ps -A”

doug — adb shell — 134×35

```
walleye:/ # ps -A | grep -v root
    USER           PID   PPID     VSZ    RSS WCHAN              ADDR S NAME
logd            618     1 10868692  4472 SyS_rt_sigsuspend    0 S logd
lmkd            619     1 10772016   940 SyS_epoll_wait       0 S lmkd
system          620     1 10774984  1996 SyS_epoll_wait       0 S servicemanager
system          621     1 10776152  2428 SyS_epoll_wait       0 S hwservicemanager
system          622     1 10774828  1832 SyS_epoll_wait       0 S vndservicemanager
system          623     1 10774460   920 do_wait              0 S qseecomd
system          633   623 10862364   412 SyS_rt_sigsuspend    0 S qseecomd
system          656     1 10827812  1868 binder_thread_read   0 S android.system.suspend@1.0-service
system          657     1 10774696   908 binder_thread_read   0 S android.hardware.atrace@1.0-service
system          659     1 10776516  1092 binder_thread_read   0 S android.hardware.gatekeeper@1.0-service-qti
system          660     1 10777436  1620 binder_thread_read   0 S android.hardware.keymaster@3.0-service-qti
system          663     1 11094156 12920 SyS_epoll_wait       0 S surfaceflinger
system          665     1 10897032  3648 binder_thread_read   0 S android.hardware.graphics.composer@2.1-service
system          666     1 10809872  1220 binder_thread_read   0 S android.hardware.configstore@1.1-service
system          667     1 10829988  1636 binder_thread_read   0 S android.hardware.graphics.allocator@2.0-service
system          703     1 10863188   884 futex_wait_queue_me  0 S time_daemon
tombstoned      830     1 10771360  1084 SyS_epoll_wait       0 S tombstoned
statsd          861     1 10879040  2048 SyS_epoll_wait       0 S statsd
system          876     1 10773976  1964 binder_thread_read   0 S android.hidl.allocator@1.0-service
audioserver     877     1    47196  3540 binder_thread_read   0 S android.hardware.audio.service
bluetooth       878     1 10865584  2536 binder_thread_read   0 S android.hardware.bluetooth@1.0-service-qti
cameraserver    879     1   126476  3060 binder_thread_read   0 S android.hardware.camera.provider@2.4-service
media           880     1    16824  1056 binder_thread_read   0 S android.hardware.cas@1.2-service
context_hub     881     1 10792416  2436 binder_thread_read   0 S android.hardware.contexthub@1.0-service
media           882     1    19260  1160 binder_thread_read   0 S android.hardware.drm@1.0-service
media           883     1 10795584  2092 binder_thread_read   0 S android.hardware.drm@1.3-service.clearkey
media           884     1    21872  1064 binder_thread_read   0 S android.hardware.drm@1.3-service.widevine
gps             885     1 10926744  2568 binder_thread_read   0 S android.hardware.gnss@1.0-service-qti
system          886     1 10775048  2500 SyS_epoll_wait       0 S android.hardware.health@2.0-service.wahoo
system          887     1 10774988  2312 binder_thread_read   0 S android.hardware.light@2.0-service
system          888     1 10774976  2336 binder_thread_read   0 S android.hardware.memtrack@1.0-service
nfc             889     1 10863772  2396 binder_thread_read   0 S android.hardware.nfc@1.1-service
system          890     1 10774744  2256 binder_thread_read   0 S android.hardware.oemlock@1.0-service
```

- list of programs currently running on the phone, there are lots!


doug — adb shell — 134×35
oxedProcessService0:13
u0_a155 15983 863 14416584 124476 SyS_epoll_wait 0 S com.google.android.apps.messaging
u0_a115 15996 863 13695100 59584 SyS_epoll_wait 0 S
u0_a155 16041 863 13873936 84048 SyS_epoll_wait 0 S com.google.android.apps.messaging:rcs
u0_a129 16098 863 13772408 66744 SyS_epoll_wait 0 S com.google.android.apps.turbo:aab
u0_a60 16145 863 13694936 65636 SyS_epoll_wait 0 S android.process.acore
u0_a180 16418 863 13918556 96444 SyS_epoll_wait 0 S com.android.systemui:screenshot
u0_a129 16459 863 13752560 69568 SyS_epoll_wait 0 S com.google.android.apps.turbo
u0_a61 16492 863 13966020 80380 SyS_epoll_wait 0 S android.process.media
u0_a131 16514 863 13905320 71572 SyS_epoll_wait 0 S com.google.android.projection.gearhead:shared
u0_a131 16558 863 13825020 71784 SyS_epoll_wait 0 S com.google.android.projection.gearhead:car
u0_a133 16632 863 13613444 58048 SyS_epoll_wait 0 S com.google.android.setupwizard:remote
u0_a171 16657 863 14681192 73816 SyS_epoll_wait 0 S com.google.android.apps.photos
u0_a135 16756 863 13682544 63584 SyS_epoll_wait 0 S com.google.process.gapps
u0_a137 16786 863 14383620 66544 SyS_epoll_wait 0 S com.google.android.GoogleCamera
u0_a184 16867 863 13807784 61584 SyS_epoll_wait 0 S com.google.android.permissioncontroller
u0_a134 16899 863 13778620 69440 SyS_epoll_wait 0 S com.google.android.euicc
system 17001 1 10774864 2984 binder_thread_read 0 S android.hardware.dumpstate@1.0-service.wahoo
u0_a107 17017 863 13819568 71796 SyS_epoll_wait 0 S com.google.android.settings.intelligence
system 17271 863 13689956 58124 SyS_epoll_wait 0 S
u0_a114 17291 863 14769836 146200 SyS_epoll_wait 0 S com.android.vending
u0_a100 17337 863 13955072 80932 SyS_epoll_wait 0 S
u0_a114 17448 863 14019140 81212 SyS_epoll_wait 0 S com.android.vending:instant_app_installer
u0_a114 17498 863 14055196 93348 SyS_epoll_wait 0 S com.android.vending:background
u0_a102 17645 863 13681964 60752 SyS_epoll_wait 0 S com.android.connectivity.metrics
u0_a126 17669 863 13945992 71340 SyS_epoll_wait 0 S com.google.android.apps.gcs
u0_a110 17747 863 14638124 110120 SyS_epoll_wait 0 S com.google.android.googlequicksearchbox
u0_a135 17787 863 14016756 91864 SyS_epoll_wait 0 S com.google.android.gms.ui
u0_a116 17878 863 13664984 60388 SyS_epoll_wait 0 S com.google.android.configupdater
u0_a119 17888 863 13654608 65580 SyS_epoll_wait 0 S
u0_a135 17947 863 14384684 161476 SyS_epoll_wait 0 S com.google.android.gms
shell 18039 1173 10771048 1740 SyS_rt_sigtimedwait 0 S sh
shell 18049 18039 10805848 1644 wait_woken 0 S su
u0_a190 18072 863 13737984 65592 SyS_epoll_wait 0 S com.topjohnwu.magisk
walleye:/ #
Messages app
Google Play store app
Google Play Services



System apps - can't be removed, permitted to do things normal apps can't

Hardware stuff (camera, sensors, phone modem, etc) + OS kernel

Pre-installed system apps

User apps

Anything you install e.g. news apps

OEM apps (Samsung, Xiaomi etc) e.g. Settings, Dialer, Keyboard

Google apps (Google Play Services, Google Play store, Maps, Youtube, Searchbar etc)

Third-party apps (Facebook, LinkedIn, Netflix, Amazon etc)

- These system apps can start and run by themselves
- Have privileged access to phone
- Can do things silently - might not have a UI, need not ask for user permission/consent

System apps are a bit like the walls of a house, user apps are the furniture, carpets etc

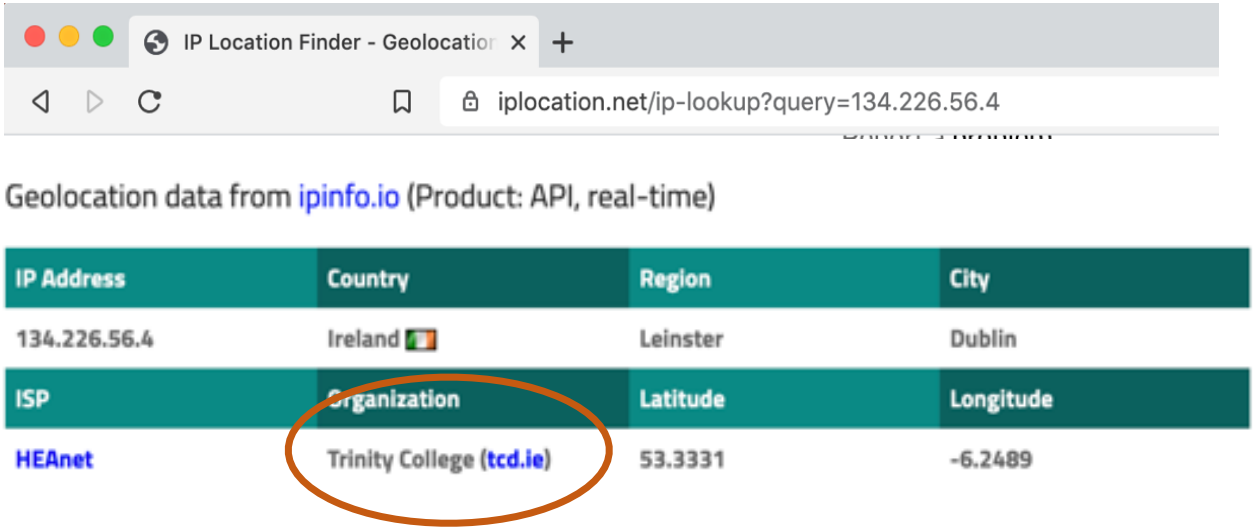
Privacy Threats: Metadata Matters

It’s not just the data, post-Snowdon we know the meta-data matters too.

- Logging app usage (app screens viewed, when and how long)
  - reveals time and duration of phone calls, send/receive of texts etc
  - effect is akin to using cookies to track user behaviour across web sites
- Identifiers sent along with data can allow linking /re-linking
  - if have user-resettable advertising id but also send handset hardware serial number alongside it in messages then easy to relink ad id to handset after reset.
  - If multiple parties gather data and tag it with the same identifier (e.g. ad id) then the data can potentially be linked.
- Logging set of installed apps can be revealing
  - Muslim prayer app, gay dating app, mental health app, what newspaper you read

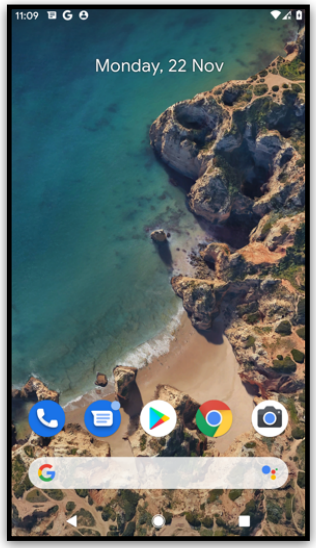
Privacy Threats: Metadata Matters

- IP address can be revealing
  - Every message sent by app to a backend server includes an IP address
  - Can map IP address to rough location (geoIP services). In city accuracy is perhaps ~2km, but can be better.

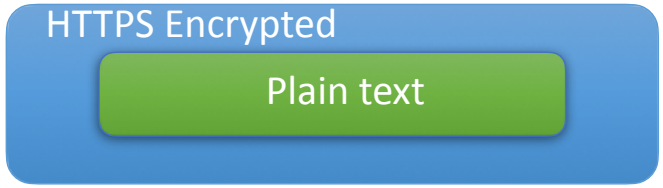


  - Handset fingerprinting

What We Did



- Setup a Raspberry Pi as WiFi AP, handset associates to AP, so we can intercept traffic
- The traffic is of course encrypted.

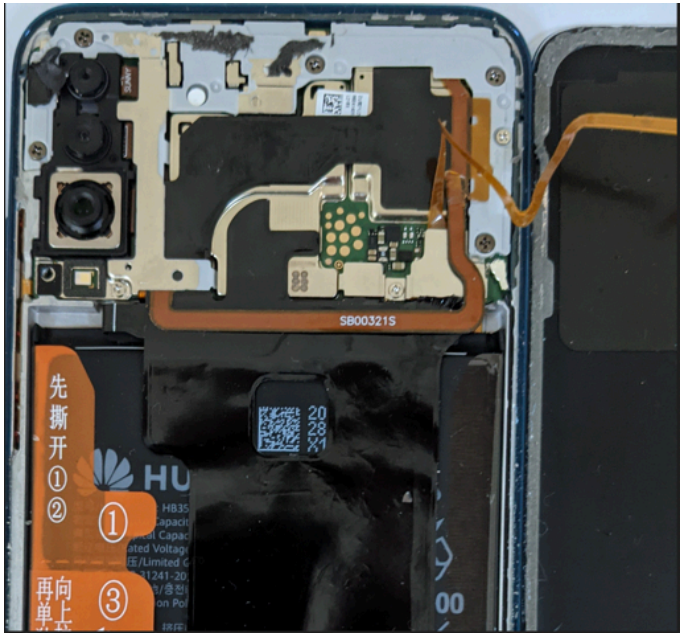


- Install mitmproxy CA cert as trusted, which requires rooting phone
- Then used Mitmproxy to intercept and decrypt the traffic sent by phone.

What We Did

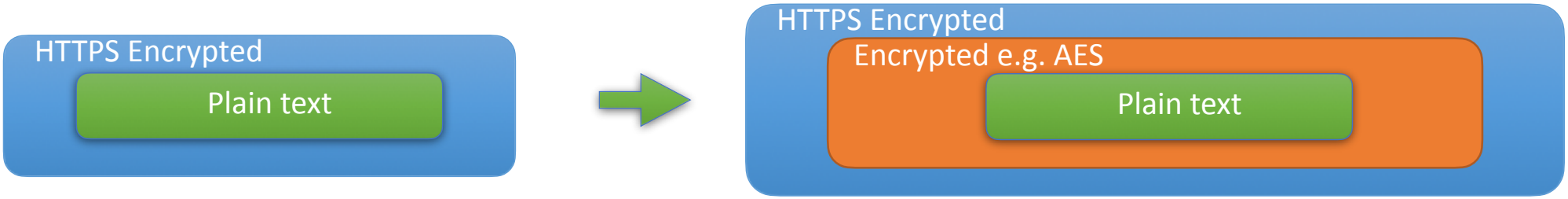
- Looked at Samsung, Xiaomi, Huawei, Realme, Google, LineageOS, e/OS handsets
  - Samsung is #1 android handset manufacturer, Xiaomi #2 (previously was Huawei)
  - e/OS is privacy-focussed open source OS, LineageOS #1 open source OS

- Some difficulties …
  - Rooting a Huawei phone is tricky (needed to use EDL/test-point mode and patch bootloader to reset unlock code).



What We Did

- Some extra difficulties …
  - Xiaomi, Realme, Huawei all use their own data encryption (in addition to https)
    - Xiaomi - Fairly sophisticated proprietary AES key exchange protocol, store keys in secure element. Key never unencrypted at rest, but can be extracted from phone memory … quite tricky
    - Realme - AES key and IV hard-coded in app, easy!
    - Huawei - Mix of AES encryption and encryption using “white-box” JNI code i.e. seriously obfuscated. Extracted AES key from memory and grabbed JNI plaintext from memory.
    - Why are the Chinese manufacturers taking these extra steps?



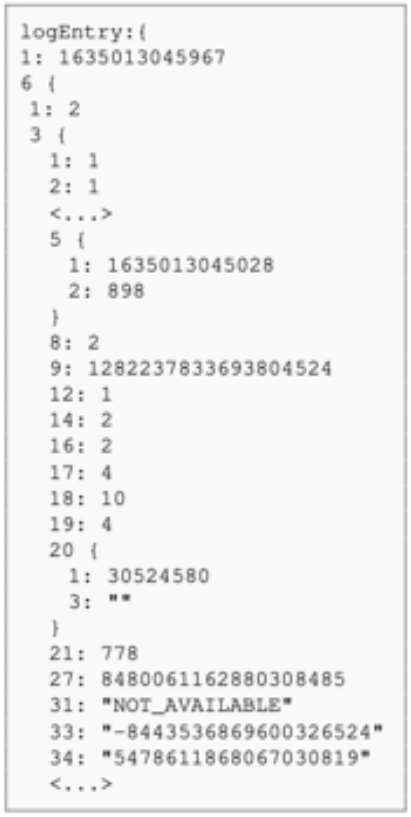
What We Did

- Wait, it's not over yet! After decryption, still need to decode binary payload.
- Sometimes its relatively easy:
  - JWT tokens, base64 and hexstring encoding, gzipping, Snappy
  - Plenty of use of Google protobufs (easy to partly decode, but often hard to interpret resulting data without schema)
- Other times harder:
  - Avro. Cannot be decoded without schema, needed to reverse engineer this from app.
  - Bond Compact Binary. Microsoft format, rubbish documentation and support. Reverse engineered schema from app, used tools to compile decoder, used decoder to deserialise and re-serialise to JSON
  - Qihoo 360 uses proprietary data format.

What We Did

```
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"
  <...>
```

What We Did



```
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"
  <...>
```

```
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: 12822378336938045248
  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"
 <...>
```

- Google Messages app (default on Huawei, Xiaomi, newer Samsung phones etc)

What We Did

Just installing a mitmproxy system cert is enough for Google handset and to view data sent by Google system apps.

… but for all the rest quite a bit of extra work is needed to view data sent.

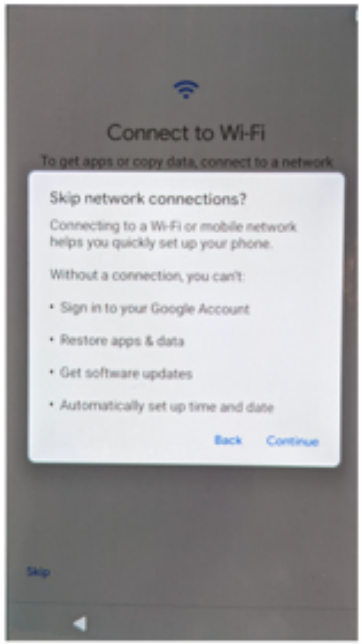
Is there some moral hazard here (do bad stuff and hide it really well)?

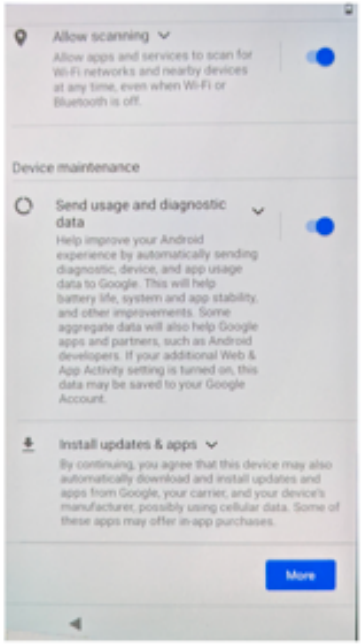
Should we have a right to view the data leaving our phones?

Claims re security concerns raised by allowing phones to be rooted are mostly nonsense - we don't lock desktops and laptops. Preventing rooting makes privacy verification v hard e.g. Apple.

Test Setup

- Mimicked a privacy conscious but busy user making minimal use of phone:
  - Factory reset a sim-free handset answered “no” to all questions asking to share data, did not login to Google (or Samsung or whatever) account.





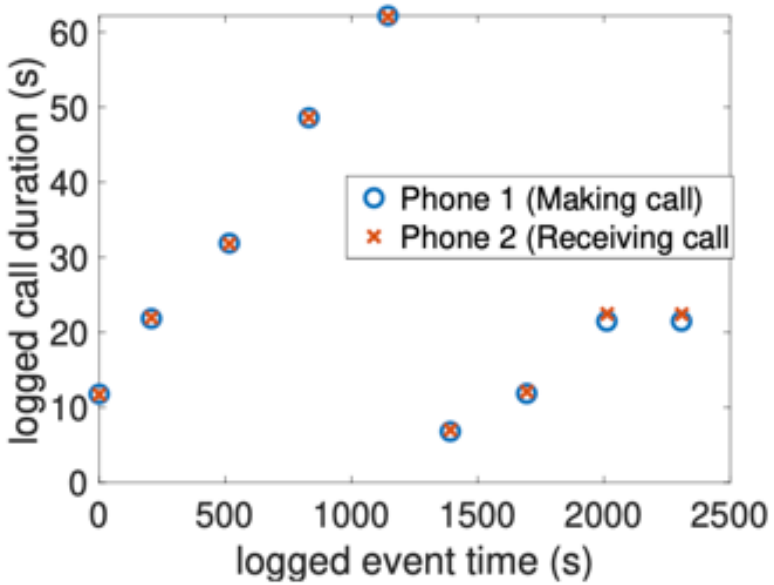
- Left phone idle, made/received a call, sent/received a text, looked at settings

What We Found - Xiaomi Is Tracking Users

```
POST https://tracking.intl.miui.com/track/v4
Headers
  OT_SID: 1904b90...536c63d4
  OT_ts: 1627029461128
  OT_net: WIFI
  OT_sender: com.miui.analytics
    "seq": [
      {
        "event": 1,
        "pkg": "com.google.android.dialer",
        "class": "com.android.incallui.InCallActivity",
        "ts": 1627028918422,
        "vn": "67.0.383690429",
        "stat": "app_start"
      },
      {
        "event": 2,
        "pkg": "com.google.android.dialer",
        "class": "com.android.incallui.InCallActivity",
        "ts": 1627028934973,
        "vn": "67.0.383690429",
        "duration": 16551,
        "stat": "app_end",
        "app_duration": 16551
      }
```

- Xiaomi analytics system app logs time history of app screen usage. Reveals e.g. time/duration of phone calls. There is no opt out.

- Time/duration of phone call is enough to discover pairs of communicating handsets:



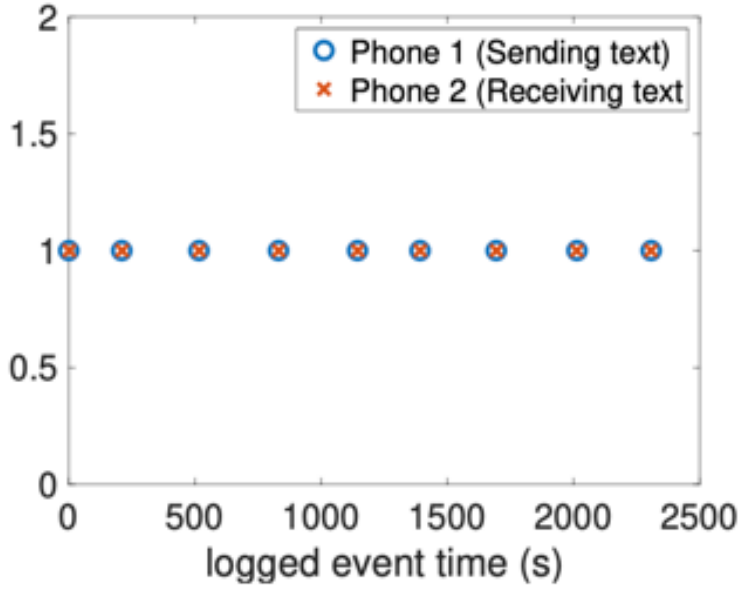
- Note: Xiaomi logs time history of app screen usage for all apps, not just dialer. Akin to cookies tracking across web sites.

What We Found - MS/SwiftKey Is Tracking Users


```
{'event': {'metadata':
{'installId': b'\xe7\x19\xec\xa8KD\xff\xa1&E\xa3\x066G\xf6[', 'appVersion': '7.8.3.5', 'timestamp':
   {
   'utcTimestamp': 1628165014657, 'utcOffsetMins': 0}, 'vectorClock': {'major': 103, 'minor': 482, 'order': 100}
   },
   'application': 'com.google.android.apps.messaging', 'durationMillis': 6891,
   'typingStats':
      {'totalTokensEntered': 0, 'tokensFlowed': 0, 'tokensPredicted':
      0, 'tokensCorrected': 0, 'tokensVerbatim': 0, 'tokensPartial': 0, 'netCharsEntered': 3, 'deletions': 1,
 'characterKeystrokes': 0, 'predictionKeystrokes': 0, 'remainderKeystrokes': 0, 'predictionSumLength': 0, 'typingDurationMillis': 837, 'emojisEntered': 0, 'totalTokensEnteredEdited': 0, 'tokensFlowedEdited': 0, 'tokensPredictedEdited': 0, 'tokensCorrectedEdited': 0, 'tokensVerbatimEdited': 0, 'tokensPartialEdited': 0},
   'languagesUsed': 0, 'termsPerLanguage': {}, 'tokensPerSource': {}, 'tokensShownPerSource': {'': 6, 'en_GB/en_GB.lm': 16, 'user/dynamic.lm': 6},
   'userHandle': 0
}}
```

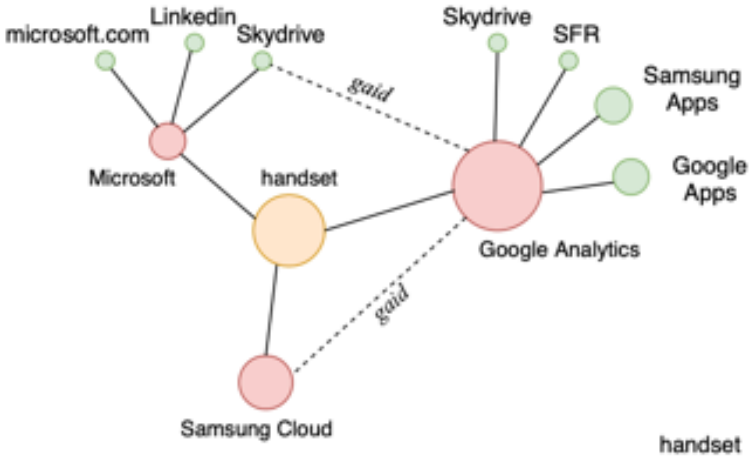

- Whenever keyboard is used within an app, details are sent to Microsoft. Reveals e.g. when messages are sent. There is no opt out.

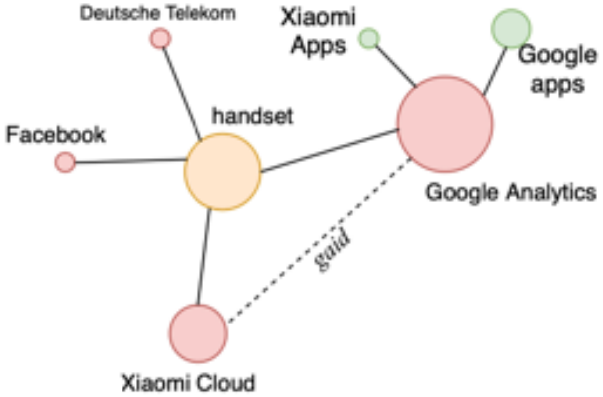
- timing of SMS messages is enough to discover pairs of communicating handsets:

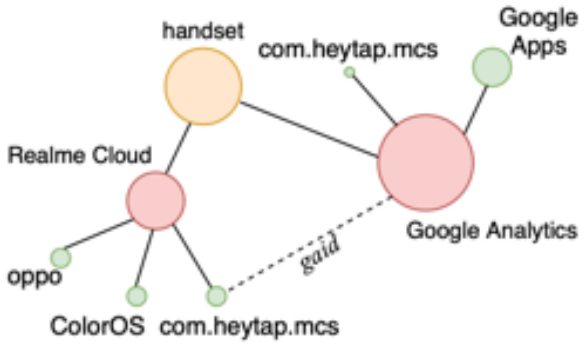


- Note: Swiftkey logs time history of app screen usage for all apps using keyboard, not just messages app.

What We Found - Data Ecosystem







What We Found - Data Ecosystem

- Pervasive use of long-lived device identifiers in messages:

	Samsung	**Xiaomi**	**Realme**	**Huawei**	**LineageOS**	**/e/OS**	**Google**
Long-lived Device Identifiers	IMEIs, hardware serial numbers	IMEIs, Secure DeviceID, MD5 hash of Wifi MAC address	IMEI, deviceID, guid	hardware serial number, device RSA cert	-	-	IMEI, hardware serial number, Wifi MAC address

- Note: Checking for system and app updates etc does not require this, could easily be done anonymously or using short-lived identifiers

What We Found - Relinkability of Ad IDs

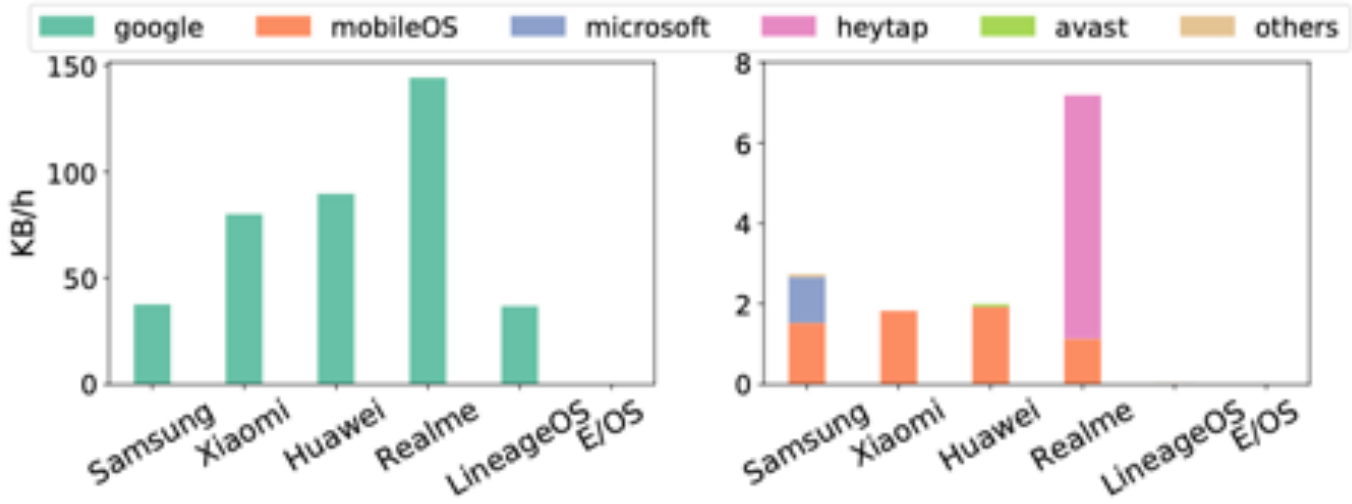
- Samsung, Xiaomi, Realme, Google (but not Huawei) send long-lived identifiers alongside ad ids.
- If use resets ad id, its trivial to relink new ad id back to handset

	Samsung	Xiaomi	Realme	Huawei	LineageOS	/e/OS	Google
Long-lived Device Identifiers	IMEIs, hardware serial numbers	IMEIs, Secure DeviceID, MD5 hash of Wifi MAC address	IMEI, deviceID, guid	hardware serial number, device RSA cert	-	-	IMEI, hardware serial number, Wifi MAC address
Resettable Identifiers Relinkable to Device	Samsung Consumer ID, Firebase IDs	VAID, Google Ad ID	VAID, OAID, device_id, registrationId, Google Ad ID, Firebase IDs	-	-	-	AndroidID, Google Ad ID

What We Found - Everyone Tracks Installed Apps

- Samsung, Xiaomi, Realme, Huawei, Google, Heytap (Realme partner) all log list of installed apps
- Note: Checking for updates etc could be done per app and without tagging message with long-lived device id.

What We Found - Google



Google collects >10x more data than anyone else, on every phone except e/OS one. But what data is being sent? Opaque and undocumented.

What We Found – Google Play Services

```
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"
  <...>
```

```
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: 12822378336938045248
  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"
 <...>
```

- Data sent is tagged with Android ID
- Sent even when “Usage & Diagnostic” option is set to off.

- Google Messages app (default on Huawei, Xiaomi, newer Samsung phones etc)

What We Found – Google Play Services

What We Found – Google Play Services

- Frequent /log/batch calls. Share information on activity of handset → includes GAEN telemetry and client app telemetry.

```
POST https://play.googleapis.com/p/log/batch
Headers:
   Authorization: Bearer ya29.a0AfH6SMCv...
   X-SERVER-TOKEN: CAESKQDyi0h8u1NFMSbtIY...
   Cookie: NID=204=cZfpA_V3EeVcgH8ON4DUR...
   User-Agent: Android/com.google.android.gms/202117028 (
walleye PPR2.180905.005); gzip
```

NID Cookie allows linkage to / checkin request data

- Sent every 10-20 minutes
- Sent even when “Usage & Diagnostic” option is set to off.
- Data sent is opaque and undocumented, there is no opt out

System apps are a bit like the walls of a house, user apps are the furniture, carpets etc

Google

Also Xiaomi, Microsoft …

Mitigations?

- F-Droid apps are verified open source and tracker free - if you can find a replacement app there its a good idea to use it
  - Simple Dialer
  - QKSMS
  - Keyboard
  - Firewall e.g. Blokada
- Brave browser or the like
- MagicEarth Maps
- Apple?
- Tell your TD that you care about privacy - enforce GDPR