Why Google Should Stop Logging Contact-Tracing Data

Published by **Joel Reardon** on April 27, 2021

Earlier, we wrote about some attacks that can be done in GAEN-based contact-tracing systems, which is the Google-Apple Exposure Notification system that is used by many countries and regions worldwide to do digitally-assisted contact tracing. Recently, we found that Google's implementation of GAEN logs crucial pieces of information to the system log, which can be read by hundreds of third-party apps and used for the privacy attacks that we previously warned about. We'll start with a nutshell description of GAEN contact tracing, then show what data is logged, and finally explain why this logging is so concerning to us. Before we begin, we want to stress that the issues we highlight here are implementation errors in the system, *not* an inherent design flaw of distributed contact tracing.

How GAEN Works

In decentralized smartphone-assisted contact tracing, a person's smartphone records information about proximate encounters with other nearby phones. This data is then used to infer that their owners were also near one another. A trusted authority (e.g., a public health department) can then reveal which of these encounters involved a contagious person, and then a mobile app can perform a risk calculation to assess a user's risk of having been infected by COVID-19.

The GAEN implementation, in particular, uses random-looking *rolling proximity identifiers* (RPIs) that are broadcast through the Bluetooth radios in a user's phone and can be heard by other nearby Bluetooth-enabled phones. These RPIs periodically change (e.g., every 15 minutes) so that a user cannot be tracked through the RPIs that they broadcast. That is, the same user will broadcast different RPIs over time.

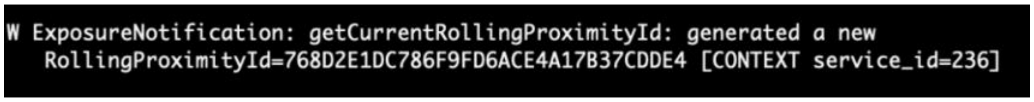
The set of RPIs that a user "*hears*" from nearby devices corresponds to the set of unique proximate encounters with other users; some of these encounters put the user at risk of contagion. GAEN helps users identify the RPIs that are "*risky*" by having their devices retrieve **all** risky RPIs, and then each user's device can assess (offline) whether their device was exposed to any of these risky RPIs. The fundamental privacy feature claimed by GAEN—in contrast to centralized systems—is that the RPIs that a user's device sees over time never leave the device. That is, users learn from a health authority the set of RPIs that were broadcast by infectious people, but no one else learns who or if anyone ever heard these RPIs.

The downloading of all "*risky*" RPIs is made efficient by using a "temporary exposure key (TEK)" to generate all the positive user's RPIs for the day (i.e., all of a user's RPIs are not actually random, but instead derived from the TEK). Thus, if you know the TEK of a positive user, you can compute all the RPIs that would have been broadcast by their device. Health authorities publish lists of the risky TEKs, which allows users, or indeed anyone, to *link* all the RPIs for an infectious user. That is, when RPIs are linkable, you may not know *who* broadcast two different ones, but you know that both came from the same person. At least, that is how the privacy guarantees are supposed to work.

What Gets Logged

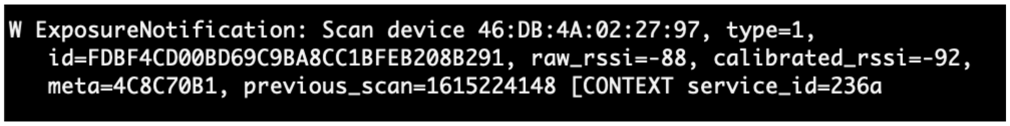
In Android, GAEN's functionality is offered to certified health authorities by Google Mobile Services (GMS), which according to Google's definition, "is a collection of Google applications and APIs that help support functionality across devices." Google's implementation of GAEN writes all RPIs—both those broadcast by the user's device *and* those observed from other users' devices—to the phone's system log. For the RPIs that are received from other devices, GMS also logs the current Bluetooth MAC address of the sending device. For privacy reasons, Android prevents apps from knowing their own MAC addresses, so it can only log the MAC addresses that are broadcast along with the

RPIs from other devices. The next three figures show (i) logging of a device's own RPI, (ii) logging of a heard RPI and MAC address, and (iii) an illustration of two devices using GAEN and the logging that happens.



```
W ExposureNotification: getCurrentRollingProximityId: generated a new
   RollingProximityId=768D2E1DC786F9FD6ACE4A17B37CDDE4 [CONTEXT service_id=236]
```

Figure 1: Example of an Android system log containing an RPI generated by the device that is about to be broadcast.



```
W ExposureNotification: Scan device 46:DB:4A:02:27:97, type=1,
   id=FDBF4CD00BD69C9BA8CC1BFEB208B291, raw_rssi=-88, calibrated_rssi=-92,
   meta=4C8C70B1, previous_scan=1615224148 [CONTEXT service_id=236a
```

Figure 2: Example of an Android system log containing an RPI linked to the randomized MAC address (46:DB:4A:02:27:97) of another nearby device that was heard.

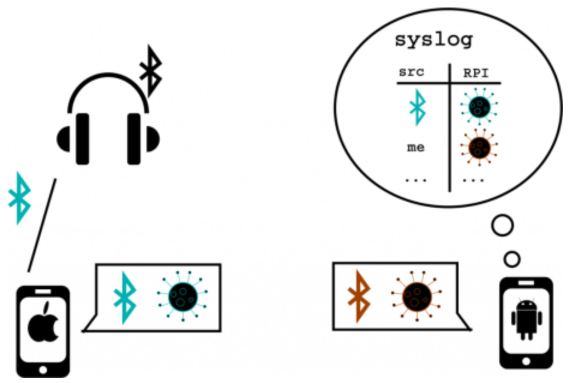
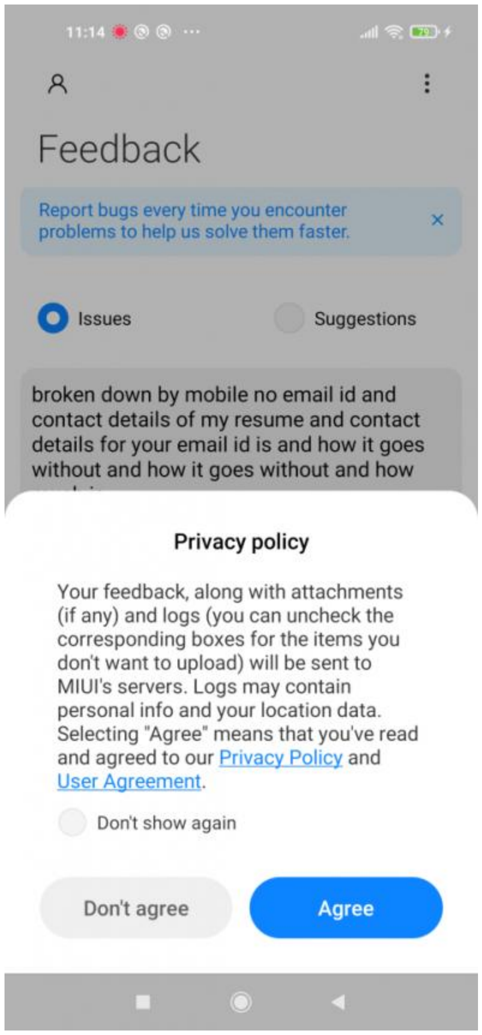


Figure 3: The GAEN system in practice. An Apple and an Android device are broadcasting RPIs along with their (current) Bluetooth MAC addresses. Devices may be also broadcasting their Bluetooth MAC addresses in other contexts, e.g., because Bluetooth is on or being used. The Android device is storing all observed MACs+RPIs to its system log, along with its own RPIs. Note that by design, Android does not let apps learn their own MAC addresses.

Since 2012, Android security controls prevent normal apps that users download from the Play Store from reading the system logs (unless there's a side or covert channel). However, Google allows phone hardware manufacturers, network operators, and their commercial partners (e.g., advertising libraries) to pre-install "privileged" apps. One feature of these privileged apps is that they have access to additional permissions that are otherwise not afforded to third-party apps downloaded from the Play Store. One such permission is READ_LOGS, which allows privileged apps to read the system logs.

As a 2020 research study showed, these pre-installed apps are highly prevalent. A stock Xiaomi Redmi Note 9, for example, has 77 pre-installed apps that we identified, 54 of which have the READ_LOGS permission. A Samsung Galaxy A11 was found to have 131 privileged apps, 89 of which had READ_LOGS. Perhaps it is worth mentioning that both Samsung's and Xiaomi's privacy policies explicitly mention collecting log data, with the latter explicitly mentioning system logs and application crash logs. (We cite these two companies only as examples, and not because we believe they are outliers.) Below is what the Redmi Note 9 looks like if a user tries to submit feedback:


11:14

Feedback

Report bugs every time you encounter problems to help us solve them faster.

Issues Suggestions

broken down by mobile no email id and contact details of my resume and contact details for your email id is and how it goes without and how it goes without and how

Privacy policy

Your feedback, along with attachments (if any) and logs (you can uncheck the corresponding boxes for the items you don't want to upload) will be sent to MIUI's servers. Logs may contain personal info and your location data. Selecting "Agree" means that you've read and agreed to our Privacy Policy and User Agreement.

Don't show again

Don't agree Agree

Figure 4: Dialog from a Xiaomi device indicates that system logs may contain personal information and will be sent.

This is not to pick on these or any OEMs: it is unlikely that they collect log data with the understanding that they are now receiving user's medical and other sensitive information as a result of Google's implementation. The key point is that one should simply **not log sensitive data** to the system log in the first place. If sensitive data is saved to the system log, one loses control over what happens to it, because it is accessible to other entities. Indeed, Google's GAEN implementation should follow Android's own Privacy Security Best Practices, which state that "logging data increases the risk of exposure of that data" and that "multiple public security incidents have occurred as a result of logging sensitive user data."

Privacy Risks

Now we want to describe the specific privacy risk for contact-tracing app users that arise from this logging. There are risks even for Apple users and Android users who do not have any app that uploads their system logs, though there are greater risks for Android users who do have their logs uploaded by these third-party apps. As well, there are some attacks that become more powerful when combined with databases from other known types of data collection.

Inferring COVID-19 Status

An entity that reads one user's logs can learn the RPIs that that particular user broadcasts and the ones they hear. When combined with the positive TEKs published by public health authorities, one can learn the user's health status, i.e., whether they have published their TEKs on account of a positive diagnosis. Based on the RPIs a user's device hears, a risk calculation can be done on that user's behalf, i.e., to function as one expects from an exposure-notification app.

An entity that collects logs can also associate it to the user's identity. Any app, including pre-installed ones, can have permissions to get the email or phone number of a device, but there are other persistent identifiers that can be accessed as well. Not only that, but when the phone boots up, the user-set device name also gets printed to the system log (along with the real Bluetooth MAC address, not the randomized one). For example, this gets printed on one of my devices:

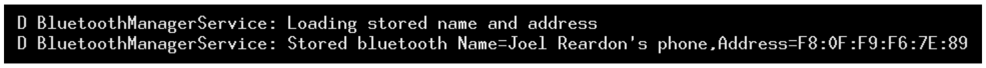
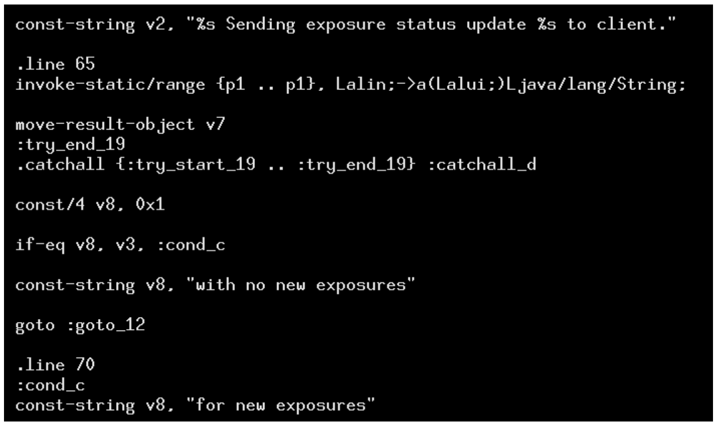

```
D BluetoothManagerService: Loading stored name and address
D BluetoothManagerService: Stored bluetooth Name=Joel Reardon's phone,Address=F8:0F:F9:F6:7E:89
```

Figure 5: Example of phone name and persistent identifier being logged.

Moreover, there may be more direct indicators about a user's health status that are logged. If there are no exposures detected, there is an "all clear" message that is printed: "Sending exposure status update with no new exposures to client." While we didn't have any exposures that would trigger the opposite behavior, looking at Google's implementation of GAEN indicates that a different message would be printed were the user at risk. Even if no message gets printed in this case, the absence of an "all clear" is indicative in itself. This further indicates that the user *is aware* that they are at risk of being contagious to others.



```
const-string v2, "%s Sending exposure status update %s to client."

.line 65
invoke-static/range {p1 .. p1}, Lalin;->a(Lalui;)Ljava/lang/String;

move-result-object v7
:try_end_19
.catchall {:try_start_19 .. :try_end_19} :catchall_d

const/4 v8, 0x1

if-eq v8, v3, :cond_c

const-string v8, "with no new exposures"

goto :goto_12

.line 70
:cond_c
const-string v8, "for new exposures"
```

Figure 6: Code from Google's implementation of GAEN that indicates whether the user is at risk for exposure.

Collecting Social Graphs

An entity that reads many user's logs can further de-anonymize the RPIs that other users' devices hear, provided those RPIs appear in multiple logs. That is, if an entity knows both Alice and Bob's RPIs, they can recognize that Bob heard one of Alice's RPIs, which allows them to learn that Alice and Bob were at the same place at a specific time. This social graph information becomes richer the more users contribute their data. Even if the entity cannot de-anonymize a particular RPI, simply knowing that Alice and Bob heard the same RPI at the same time helps build this social graph.

The linking of MAC addresses to RPIs creates more problems. First, some context: MAC addresses are fixed serial numbers for networking hardware that are used in low-level communication, but for mobile phones, they are now periodically randomized to a new unlinkable value. MAC addresses must be broadcast when using Bluetooth, its simply how low-level networking works, and so randomizing it prevents someone from being tracked because they broadcast the same MAC address everywhere they go. In the context of GAEN, both the MAC address and the RPI change in tandem.

Inferring Location Trails

Given that MAC addresses are random, storing them alongside the RPI may seem superficially harmless. Linking these two values, however, is problematic, because the Bluetooth MAC address is used in other contexts. For example, there are databases that link Bluetooth MAC addresses to the physical location they were observed. Wigle is a public service that offers this based on a volunteer effort, and both Facebook and Instagram are apps with more than a billion installations and have privacy policies stating that they are collecting exactly this information. Such gathered information is trivially associated with a persistent identifier for a particular device. The next figure shows the two data collectors: the left is the Bluetooth MAC address to location database that is already being curated; the right is what can be learned by any app that uploads a user's system log.

Figure 7: Two databases that can be collected. The left database is curated by devices scanning for Bluetooth devices and reporting the MAC address and the location where they are observed. Such collection can include a persistent identifier, such as the user's advertising ID, Android ID, or IDFA that links who observed different Bluetooth devices, indicated by the observer column. The right database consists of log data collected by apps reading the system log. This collection may be accidental, but it results in the linking of a user, the RPIs they broadcast, and the MACs+RPIs that they heard.

An entity that collects users logs can turn the RPI they hear into the corresponding MAC address; with access to existing databases, they can turn the MAC address into a geolocation. This allows them to learn a location history of a user based on geolocating the RPIs they hear. If a user links their RPIs by publishing their TEK (i.e., they report a positive test and upload their data), then they reveal their location history whenever they broadcast their RPI to an Android device nearby that uploads their logs. That is, iPhone users are just as much at risk of this occurring, because the necessary data collection occurs entirely by other devices. The next figure shows the result of a hypothetical database join of the previous figure, which links location to RPI through MAC address and RPI to broadcaster or observer in the context of a social graph.

Figure 8: A hypothetical join of the two example databases. The first row shows what is known about the Apple user: its MAC+RPI is tied to location, and two others become linked for having seen it. The next two rows show how the magenta user has their MAC address collected in a Bluetooth collection capacity, but not through log access; therefore the RPI and the MAC do not become linked. The final row shows the worst case concern: a user has their own RPI reported by themselves (tied to their identity) and by someone else (tied to their MAC address), which is in turn linked to location by Bluetooth collection.

Responsible Disclosure and Remediation

We responsibly disclosed this issue to Google on February 19th, 2021. As of this writing, however, it remains unresolved. As more than 60 days have elapsed, we are following Google's recommendation that researchers publish their findings about the vulnerability. Even if a patch were released to stop this logging, however, log data may have been already uploaded. It is crucial that any entity that has collected system log data from Android devices sanitize any entry containing contact-tracing data, and that this unnecessary logging be stopped as soon as possible.

To reiterate our initial point: this is an implementation issue, and not a fundamental problem with the underlying framework. After Google rejected our bug bounty submission and failed to fully acknowledge or fix the issue, we had a long debate about the ethics of making this public. Ultimately, we felt that making this public is not putting consumers any more at risk and will hopefully lead to a fix, accountability, and improved privacy protections for consumers. To be absolutely clear: this is a fixable problem in the implementation and should absolutely not be used by charlatans and opportunists to further undermine public health efforts. Wear a mask and get vaccinated!

Acknowledgements

This research is funded by the U.S Department of Homeland Security Science and Technology Directorate (DHS S&T) under OTA 70RSAT21T00000003. Any opinions contained herein are those of the author(s) and do not necessarily reflect those of DHS S&T. This research and writing was done along with Serge Egelman, Narseo Vallina-Rodriguez, and Nathan Good, with additional contributions from Robert Richter and Julien Gamba.

Categories: UNCATEGORIZED

Joel Reardon

Dr. Joel Reardon is the Forensics Lead and Co-Founder of AppCensus. Dr. Reardon is also a professor at the University of Calgary and a world-renowned expert in digital forensics and security.

Related Posts

The Curious Case of Coulus Coelib

The technique we invented to uncover the active exploitation of side and covert channels continues to work. One such side channel comes from an obscure, purportedly-Panamanian company. A look at their code makes for an Read more…

Side Channel Exploited by My2022 App

Recently, Canada's Citizen Lab released a detailed investigation of the mobile app provided by the Government of the People's Republic of China that appears mandatory for all attendees of the upcoming Winter Olympic games. We took a look to see if there was anything else we could find (version 2.0.4, released January 2022), and noticed evidence of a side channel being exploited. In particular, we found that the MAC address of the mobile device is being sent to the domain "bigdata.beijing2022.cn".

What the Huq?

Huq Industries is a UK company that provides location-based services and mobility data, some of which is collected via an SDK that is integrated with various apps. From Huq's marketing materials: "The most accurate grade Read more…