Linkability of Rolling Proximity Identifiers in Google's Implementation of the Exposure Notification System

Zak Brighton-Knight
School of Computing, The Australian National University

Jim Mussared
George Robotics

Alwen Tiu
School of Computing, The Australian National University

Created: 23 September 2020
Reported to Google: 20 October 2020
Last updated: 21 March 2021

Description of the vulnerability

Google's implementation of the Google/Apple Exposure Notification (GAEN) System appears to use a resolvable Random Private Address (RPA) to advertise the Rolling Proximity Identifiers (RPI), instead of a non-resolvable RPA whose use is specified in its own advertised specification (see the 'Broadcasting Behavior' section in that document).

An attacker who possesses the Identity Resolving Key (IRK) of the bluetooth adapter of a target phone will be able to determine whether a given RPA belongs to that phone. A consequence of the GAEN System advertising RPIs using resolvable RPAs is that an attacker observing an RPI X advertised on an RPA Y will be able to determine whether the RPI X belongs to the target phone by checking whether the address Y can be resolved using the IRK of the phone. As long as the IRK of the phone does not change (which is the case in Android phones, unless the phone is factory-reset), the attacker will be able to associate an observed pair of RPI/RPA with that IRK, irrespective of when that observation is made, and independently of any other observations.

On its own, this GAEN vulnerability is not very interesting, as the IRK of a phone is not usually revealed to another device unless the phone is trying to pair with the device. However, there are several exploits that would allow an attacker to pair with a phone silently, to perform an IRK theft. Chaining these exploits with the above exploit would then allow a de-anonymization attack on a device running Google GAEN API.

Disclosure

We first observed the use of resolvable RPAs in GAEN in early September 2020, when we were investigating the RPA/RPI out-of-sync rotation for some phone models.[1] We reported this bug to Google on Oct 20th, 2020, after it became clear that this could be chained with a number related vulnerabilities we discovered to perform a de-anonymization attack silently. In particular three of these related vulnerabilities are (1) identity address leakage in Oppo phones (reported to Oppo on Sept 18th, 2020), (2) silent pairing issues in some older Samsung phone models (reported to Samsung on Sept 20th, 2020) and (3) identity address leakage by Xiaomi ShareMe app (reported to Xiami on Oct 24th, 2020). None of these issues were resolved at the time of writing.

A Quick Demo

The following short video shows how the GAEN vulnerability mentioned above can be exploited, by chaining it with another vulnerability, to link the RPIs broadcast by a vulnerable phone (Oppo Reno2 Z).

https://youtu.be/kljBdOfegz0

This video shows the most severe case of the attack chain, where the identity address and the IRK are obtained stealthily, and thus the RPIs broadcast by the victim's phone are completely de-anonymized, without any interaction from the user of the phone.

The exploit shown in the video uses a modified version of the BlueZ btmgmt tool. The only difference between the modified btmgmt and the official version is that the modified version outputs more information, so it is easier to demonstrate the entire attack chain within one tool rather than running several tools. This modified version of btmgmt can be found here:

https://github.com/alwentiu/btmgmt-alt

[1] Reported separately here: https://github.com/alwentiu/exposure_notification

Reproducing the Vulnerability

The fact that Google's GAEN API advertises using resolvable RPAs is trivial to observe. Simply run a BLE scanner to identify the EN broadcasts, which are tied to the service UUID FD6F. Here is an example frame obtained from a BLE scan for this service UUID, captured using the btmon program in linux:

```
> HCI Event: LE Meta Event (0x3e) plen 40                  #80 [hci0] 13.972823
      LE Advertising Report (0x02)
        Num reports: 1
        Event type: Scannable undirected - ADV_SCAN_IND (0x02)
        Address type: Random (0x01)
        Address: 47:5F:08:E7:BB:C9 (Resolvable)
        Data length: 28
        16-bit Service UUIDs (complete): 1 entry
          Unknown (0xfd6f)
        Service Data (UUID 0xfd6f): a09a33d02f18bf5fca46ac9efffcb57e244b8f2a
        RSSI: -76 dBm (0xb4)
```

The highlighted part shows the random private address, which is labelled as 'Resolvable' by btmon. This has been observed in several Android phone models we have tested, including Google Pixel 2 XL (Android 11), Google Pixel 4 XL (Android 10), Samsung Galaxy S20 (Android 10), and Oppo Reno2 Z (ColorOS 7.1/Android 10), Samsung Galaxy S8 (Android 9), Samsung Galaxy S7 Edge (Android 8), Samsung Galaxy Note 5 (Android 7).

What's more interesting is how this can be chained with a variety of exploitation methods to de-anonymize RPIs. But if Google fixes this vulnerability (by advertising the EN over non-resolvable RPA, as required by the EN specification), the RPI linkage will be harder to establish even if the bluetooth identity address of the target phone has been compromised. However, Google's engineers have confirmed that the fix will likely require a change to the Android Platform.

Exploitation Methods

On its own, the above GAEN vulnerability is not exploitable, since the attacker would need the IRK to perform the attack, and IRK is not normally available without pairing with the phone. However, there exist several ways for the attacker to obtain the IRK, silently without user interaction. We show here the methods we have discovered. These are not the only methods to obtain the IRK; we omit methods that require user interaction when the attack is launched (e.g., through social engineering), though some methods may require the presence of certain commonly used apps or features in the target phone (in particular, bluetooth-related sharing features from various vendors).

Method 1: Obtaining the IRK through the Identity Address

This may sound silly, as the Identity Address of a BLE device is not usually publicly available, unless the device is in a 'discoverable mode' (e.g., when it is trying to actively pair with another device), but we'll see later that there are at least a couple of ways of obtaining the identity address stealthily.

Once the identity address is known, the attacker can detect the presence of the device, if its bluetooth is on, using L2CAP echo requests (this can be done using, e.g., l2ping in linux). But there is no way to associate an RPA to the identity address without knowing the IRK.

There are actually three attack methods in this category, corresponding to three different ways of getting the identity address. But before we explain these three methods, first let us explain how one can ***obtain the IRK through the identity address of the phone, silently***, without any visible indication shown in the target phone. This exploits a feature introduced in Bluetooth 4.2 onwards, called the cross-transport key derivation (CTKD) protocol. A vulnerability related to this protocol, called BLURtooth, has recently been disclosed. However, the BLURtooth exploit is concerned with long term key overwrite, which has been addressed in Bluetooth Core Specification 5.2. What our exploit aims to achieve is something milder, which is to obtain the IRK silently, and does not require long term key overwrite (hence the mitigations for BLURtooth may not be sufficient to prevent our exploit). We believe this exploit was not known publicly at the time we discovered it.

To obtain the IRK from a remote bluetooth device, a pairing process needs to be initiated, either by the remote device or the local device. When connecting through BR/EDR, it appears that the default behaviour, when there was no prior established pairing between the devices, is that a Secure Simple Pairing will be initiated. The Bluetooth Core Specification 5.2 (Vol 3, Part C, 5.2.2.3 'Simple Pairing after authentication failure') states that:

> "When both devices support Secure Simple Pairing all non-SDP connections are encrypted regardless of whether security was required or whether the devices are bonded or not. The initial connection between the two devices will result in a link key through Secure Simple Pairing. "

In our tests, we could trigger the pairing process by either performing a GATT connection over BR/EDR, or issuing a security request directly. After the security request is initiated, the devices exchange IO capabilities, which are then supposedly used to determine the authentication algorithm, e.g., whether or not the user of the (local/remote) device is asked to confirm the pairing (see Bluetooth Core Specification 5.2, Vol. 3, Part C., page 1317).

Based on our examinations of the bluetooth logs of the attacking device and the target phone, it would appear that the target phone's IO capability is always set to 'DisplayYesNo'. In principle, this should have triggered a pairing request confirmation to be displayed to the user of the

phone. However, the Bluetooth Core Specification 4.2 and later (including v5.2) has two provisions for when the pairing confirmation is automatically sent without user interaction:

> "The Host may allow the Link Manager to ignore the IO capabilities and use the Numeric Comparison protocol with automatic accept by setting the Authentication_Requirements parameter to one of the MITM Protection Not Required options." *(Bluetooth Core Specification 5.2, Section 5.2.2.5, page 1316)*
>
> "Second, if neither device has received OOB authentication data and if both devices have set the Authentication_Requirements parameter to one of the MITM Protection Not Required options, authentication stage 1 shall function as if both devices set their IO capabilities to DisplayOnly (e.g., Numeric comparison with automatic confirmation on both devices)." *(Bluetooth Core Specification 5.2, Section 5.2.2.6, page 1316)*

Thus if the attacker device indicates that the pairing request does not require MITM (man-in-the-middle protection) protection, the pairing would proceed silently i.e., it would use the 'Just Works' pairing method.

The above steps will allow the two devices to establish an unauthenticated link key silently. However, the Simple Pairing over BR/EDR does not exchange the IRK explicitly. To obtain the IRK, we need another exploit that relies on the Cross-Transport Key Derivation (CTKD) feature introduced in Bluetooth 4.2.

CTKD allows dual mode devices (i.e., devices which support both BR/EDR and LE transport) to do pairing once on either transport and derive the long term key for the other transport without needing to pair twice. Assuming the target device is a dual mode device, the attacker can use a dual mode device to connect to the target, force an unauthenticated pairing over BR/EDR. This will then trigger the CTKD. In particular, both devices will perform some steps for LE Secure Connection pairing (as part of the Security Manager Protocol (SMP)) over the L2CAP connection over BR/EDR. These steps include the derivation of the (unauthenticated) long term keys (LTKs) and the distribution of the IRK, as indicated in the Phase 3 of the LE Secure Connection pairing (see Vol 3. Part H of Core Specification 5.2).

Concretely the silent IRK theft through the identity address can be done using off-the-shelf tools, e.g., those already included in the BlueZ tools that are part of the standard Ubuntu distributions. The two main ingredients are:

- the attacker's device must indicate that MITM protection is not required and allowing the pairing to be automatically accepted, and
- the attacker's device must be a dual-mode device, supporting Bluetooth 4.2 or later,

Each of the following options assumes that the commands are being run from a recent Ubuntu OS (tested on Ubuntu 18.04 and Ubuntu 20.04), on a laptop equipped with a dual-mode bluetooth adapter supporting Bluetooth 4.2 or higher.

Using btmgmt. Using btmgmt, we need to change the bluetooth adapter settings as follows: SSP enabled, Secure Connections enabled, BR/EDR transport enabled, LE transport enabled, and Bondable flag disabled:

```
btmgmt ssp on
btmgmt bredr on
btmgmt le on
btmgmt bondable off
btmgmt sc on
btmgmt pair -c 0x03 aa:bb:cc:dd:ee:ff
```

(replace aa:bb:cc:dd:ee:ff with the identity address of the target phone).

Disabling the bondable option causes the MITM protection to be disabled. The last command indicates the local device capability as 'No Input No Output', which will force the pairing to be automatically accepted, hence the pairing is to be done using Just Works, resulting in an unauthenticated link key. The 'sc on' part turns on the Secure Connections pairing process that triggers the CTKD. This exploit is essentially what was done in the demo video mentioned earlier in this document (though we did not show the steps to turn on/off various options).

Using gatttool. Another possible exploitation technique uses BlueZ gatttool, to create a BR/EDR Secure Connections to the target phone. This does not require an explicit pairing command. The command to run is as follows:

```
gatttool -b aa:bb:cc:dd:ee:ff  --psm=31 -I
```

This will take you to the gatttool prompt. Then you just need to type 'connect' to initiate the connection. As part of the connection establishment, secure simple pairing is initiated and IRKs exchanged. Here is a video showing how this gatttool method works in extracting IRK from Pixel 4XL, running Android 10.

https://youtu.be/jjr-D0acVEQ

We now describe three ways with which we can obtain the identity address.

Method 1.a: exploiting the identity address leakage

We have observed that for at least one phone model (Oppo Reno2 Z), the bluetooth adapter of the phone is constantly in a discoverable mode whenever bluetooth is on, regardless of whether the phone is locked or whether the screen is off. So a simple inquiry scan using, e.g., BlueZ's 'hcitool scan', will reveal the bluetooth identity address of the phone.

We believe this issue may be more widespread, based on the identity leakage from a number of phones from this particular manufacturer observed from sporadic bluetooth scans in public places. Our scans also detected a number of phones from other manufacturers such as Xiaomi

and Huawei. However, without access to those phone models, it is impossible to tell what causes the identity address leakage in those phones.

We have reported this issue to Oppo on September 18th, 2020, but they have essentially flagged this as a non-issue and the bug status has been marked 'Ignored' in Oppo Security Response Center, as at Oct 18th, 2020.

Method 1.b: exploiting file sharing apps

The ShareMe app from Xiaomi has a bug that, once a phone activates the 'Receive File' functionality (even if the phone does not actually receive any file transfer), the bluetooth adapter of the phone will be stuck in a discoverable mode. This means that the phone will respond to an inquiry scan over BR/EDR, revealing its identity address. The discoverable mode persists even after the ShareMe app is closed and force-stopped. The only ways to stop the discovery mode are to turn the bluetooth off and on, or restart the phone.

We are still investigating the cause for this bug, whether this is an Android bug (triggered by an unusual way in which the ShareMe app uses bluetooth), or whether this is a bug in the ShareMe app itself. We can confirm that the bug does not manifest when the ShareMe app is run on Android 11 -- it seems that the 'Receive' functionality of the app fails to put bluetooth in a discoverable mode. We have filed a bug report with Xiaomi on October 24th, 2020. Xiaomi's developers confirmed the bug, on October 30th, 2020, and claimed that it has also been identified internally by their engineers.[2] There is no timeline given for the fix. The latest version of the app at the time of writing (version 1.29.9 -- updated on 9 March 2021) still has the bug.

This is potentially very significant since this file sharing app is widely used (Google Play Store indicates there were more than 500 million downloads).

Some built-in file sharing services, e.g., Google Nearby Share, puts the phone's bluetooth in a discoverable mode when receiving a file, so a passive (opportunistic) attack may also allow an attacker to obtain the bluetooth identity addresses, by simply running a bluetooth scan continuously.

Method 1.c: brute forcing identity addresses

Many vendors provide their own file sharing functionality, e.g., Samsung's Quick Share (Android 10 and above), Huawei Share (available for most recent Huawei phones), Oppo Share, and Google Nearby Share.

[2] This was reported by the third author of this document to Xiaomi via the Hackerone platform. A Xiaomi engineer marked the bug as 'duplicate' claiming it has been "discovered by our internal engineer", and "[T]he screenshot of detailed information has been attached , pls check", except that there was no screenshot attached. Subsequent questions by the reporter were ignored. It was not clear whether the said engineer actually reproduced the bug.

We found that Huawei Share advertises a GATT service containing the Wifi MAC address in plaintext. Since Wifi MAC and Bluetooth MAC of the same phone usually share a prefix (which could be up to 5 bytes in common), it is easy to brute force the bluetooth identity address using the Wifi MAC address. More generally, an attack that reveals the Wifi MAC address of a phone may also be used to brute force the bluetooth address of the phone.

Some versions of Oppo (e.g., Color OS 6.1) leak the bluetooth identity address in plaintext when Oppo Share is enabled -- through a GATT characteristic. In Color OS 7, this is fixed, instead of sharing the identity address in plaintext, it seems to have encrypted it.

Generally, the GATT advertisements for these sharing functions are activated only when the user is actively sharing a file. However, we found that if the phone is running another app which advertises other services over GATT (e.g., the COVIDSafe app), the sharing service will also appear. In the case of Huawei Share, we can also trick the phone to start advertising by advertising a similar service pretending to be a Huawei device (just like the Nearby Sharing trick, described next).

Method 2: Obtaining the IRK through a silent pairing bug in some Samsung mobile devices (CVE-2020-35693)

Samsung mobile devices running Android 7.1.1 or an earlier version of Android are vulnerable to a `silent pairing' attack. If a vulnerable Samsung phone has a connectable GATT service, the attacker can simply connect to it and issue a pairing command. The pairing will succeed without any user interaction, and the target phone and the attacker device will be bonded -- hence the IRK of the phone will be sent to the attacker.

This bug was reported to Samsung on Sep 20th, 2020. We received a response on Oct 21st, 2020, that Samsung considers this feature as 'working as intended'. We assume the bug will not be fixed. This bug is now registered as CVE-2020-35693. More details of this bug can be found here: https://github.com/alwentiu/contact-tracing-research/blob/main/samsung.pdf

Affected devices include the following models:

- Samsung Galaxy Note 5
- Samsung Galaxy S6 Edge
- Samsung Galaxy A3
- Samsung Tab A (2017)
- Samsung Galaxy J2 Pro (2018)
- Samsung Galaxy Note 4

To exploit this bug, the target phone needs to run a connectable GATT advertisement. In some scenarios this could be quite natural. We consider two such scenarios below.

Exploiting connection-based contact tracing apps
One plausible scenario could be someone who installs two different contact tracing apps, one based on GAEN, and the other based on non-GAEN, but bluetooth based, contact-tracing apps such as Australia COVIDSafe, Singapore TraceTogether or France's TousAntiCovid. The latter apps all run a connectable GATT service to exchange some temporary identifiers for contact tracing. This is a plausible scenario for those who need to travel between countries that use these different apps, e.g., travellers between Australia (COVIDSafe) and New Zealand (NZ Covid Tracer), or between France (TousAntiCovid) and Germany (Corona-Warn-App). It is conceivable that a person who needs to cross the border often will install both apps in their phone, and if the phone is one of the vulnerable models, it can be exploited by the attacker to obtain the IRK.

Exploiting Google Nearby Sharing
Even if the target phone does not run another contact tracing app that continuously advertises a connectable GATT service, it may be possible to trigger the phone to launch a file sharing app. We found out that recently Google Nearby Sharing has been available for many phone models and Android versions.

We found that, if the target phone has Nearby Sharing enabled and is unlocked and the screen is on, then it can be tricked into creating a connectable advertisement in response to another Nearby device running. So the attacker could pretend to be such a Nearby device to trick the phone into advertising a connectable service, and launch the attack.

Here is a video showing how to pair silently with a Samsung phone, exploiting the Google Nearby Sharing to trigger at GATT connection.

https://youtu.be/4Z1SR_6SALE

As a successful pairing has occurred, the attacker now has the IRK associated with the Samsung phone and can use it to resolve the RPA as described at the start of the document.

Method 3: Exploiting CVE-2020-12856 to obtain the IRK

CVE-2020-12856 exploits an app acting in a central role in a Bluetooth GATT connection; when the central reads an authenticated characteristic on the remote device, if there is no prior pairing relationship with the remote device, then the central will initiate a pairing request. The adversary in this case acts in the peripheral role, waiting for connections from vulnerable apps, and then returns an insufficient authentication message when the central requests a read or a write on a characteristic. The central then responds by initiating a pairing request that once complete means the peripheral (attacker) has the IRK.

Although this CVE was initially reported for Australia's contact tracing app COVIDSafe, there are still potentially other apps that are similarly vulnerable; for example our initial analysis of an early version of Singapore's TraceTogether[3] shows it is potentially vulnerable.

Privacy Implication

Obviously, there's the tracking issue that's independent of GAEN; once the bluetooth identity address of a phone is known, an attacker can use l2ping to detect the presence of the phone. We focus on the privacy implication specific to EN. Linkability of RPIs to the identity address/IRK potentially allows an adversary to link together different EN diagnosis keys, beyond the 24 hour limit required by the GAEN specification. This makes it easier for the attacker to track the infection status of victims. The de-anonymization attack shown here clearly invalidates the following claim from the GAEN Cryptography Specification

> "When reporting Diagnosis Keys, the correlation of Rolling Proximity Identifiers by others is limited to 24 hour periods due to the use of Temporary Exposure Keys that change daily. "

Mitigation

An obvious mitigation would be to modify the EN advertising so that it uses non-resolvable RPAs. However, currently it is not clear whether the Android BLE advertising API allows one to specify the bluetooth address type used for BLE advertisements. From the relevant code snippets of EN API, it seems that the parameters for EN advertisements are set through the AdvertisingSetParameters class and there are currently no options for specifying the address types. Google engineers that we corresponded to said that this would likely require a platform change, and it is not clear when this bug will be fixed.

Acknowledgement

We thank Daniele Antonioli, Chris Culnane and Vanessa Teague for their valuable comments related to this research. We thank Google for issuing a VRP reward for this research, which has been entirely donated to a charity supporting COVID-19 recovery effort.

Contact

For further information on this report, please email alwen.tiu@anu.edu.au.

[3] Our analysis of TraceTogether is based on the open source version available from https://github.com/opentrace-community. We did not analyse the actual TraceTogether app, as its terms of use (https://www.tracetogether.gov.sg/common/terms-of-use) seems to imply that any attempt to reverse engineer the app would be illegal.