



No need to ask the Android: Bluetooth-Low-Energy scanning without the location permission

Vincent Toubiana, Mathieu Cunche

► To cite this version:

Vincent Toubiana, Mathieu Cunche. No need to ask the Android: Bluetooth-Low-Energy scanning without the location permission. WiSec 2021 - 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks, Jun 2021, Abu Dhabi, United Arab Emirates. pp.1-6, 10.1145/3448300.3467824 . hal-03265556

HAL Id: hal-03265556

<https://inria.hal.science/hal-03265556>

Submitted on 21 Jun 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

No need to ask the Android: Bluetooth-Low-Energy scanning without the location permission

Vincent Toubiana *
CNIL
France
vtoubiana@cnil.fr

Mathieu Cunche
INSA-Lyon, Inria, University of Lyon CITI, Lab.
France
mathieu.cunche@insa-lyon.fr

ABSTRACT

Bluetooth-Low-Energy (BLE) scanning can be misused by applications to determine a device location. In order to prevent unconsented location tracking by applications, Android conditions the use of some BLE functions to the prior obtention of the location permission and the activation of the location setting. In this paper, we detail a vulnerability that allows applications to perform BLE scans without the location permission. We present another flaw allowing to bypass the active location requirement. Together those flaws allow an application to fully circumvent the location restrictions applying to BLE scanning. The presented vulnerability affects devices running Android 6 up to 11 and could be misused by application developers to track the location of users. This vulnerability has been disclosed to Google and assigned the CVE-2021-0328.

CCS CONCEPTS

• **Security and privacy** → **Mobile platform security**; **Access control**.

KEYWORDS

Location, Bluetooth, Bluetooth-Low-Energy, Scanning, Permission, Android, Vulnerability

1 INTRODUCTION

Although the primary use of Bluetooth-Low-Energy (BLE) is to allow the wireless exchange of data with nearby devices (headset, smartwatch, etc.), it can also be leveraged to geolocate smartphones users. Hence, some companies are deploying BLE beacons for geolocation purposes in stores [8] and results of wireless network scans are already used to locate users [23, 27]. To prevent users from being geolocated without their knowledge, Google has integrated protections in Android to prevent unconsented location tracking. Since Android 6, in order to be able to use BLE to scan for nearby devices, an application must first obtain the location permission [2]. This verification ensures that applications cannot misuse BLE to illegitimately obtain the location of a user [1]. In addition, BLE scanning requires the location to be activated on the smartphone, thus providing hint to the user that its location may be determined.

We found that the restriction on BLE functionalities can easily be circumvented on many Android devices due to an inconsistent enforcement. By abusing some features and parameters of the BLE API, a malicious application could perform a BLE scan even when it has not been granted the location permission and without activating the location setting. This paper makes the following contributions:

- we present a detailed review of the location-related restriction applying to the BLE features on Android.
- we introduce and demonstrate two flaws allowing a malicious application to bypass the location permission requirement, as a well as the active location requirement.
- we analyze the impact of these flaws on mobile application and discuss how they can be fixed.

The paper is organized as follows: Section 2 presents BLE scanning and location features of Android. Section 3 details the restrictions applying to BLE scanning features. Section 4 presents how BLE restrictions can be bypassed. Section 5 and 6 respectively look at how this flaw impacts existing Android applications and how it can be fixed. Section 7 presents the related works.

2 BACKGROUND

2.1 BLE scanning on Android

Android includes since version 4.3 an API for the support of Bluetooth Low Energy. It features mechanisms for the scanning of nearby BLE devices. The scanning features are provided by the `BluetoothLeScanner` class, in which are defined among others, essential elements for BLE scanning: the `startScan()` method and the `ScanCallback()` object and its `onScanResult()` method.

The `startScan()` method. is used to initiate the BLE scanning process and takes as parameter a `ScanCallback` object through which the results of the scan will be returned.

The `startScan` method can take additional parameters to specify the settings of the scan (`ScanSettings` object) or to specify a filter (`ScanFilter` object – see section 2.1.1).

The `onScanResult()` method. belongs to the `ScanCallback` object that is passed to the `startScan()` method when a scan is initiated. This method specifies how scan results are handled and it needs to be implemented to define this result handling process. This callback method is called whenever a new scan result is available.

2.1.1 Filtered scans. By default, all discovered BLE devices will be reported as scan results to the scan callback. It is possible to select which scan results will be reported by using a `ScanFilter` that specifies a set of matching criteria. Using this feature, an application can specify filters for services UUIDs, MAC addresses, device names or manufacturer specific data. For instance, contact tracing applications have defined a service UUID that is used to listen only to devices that are running the same contact tracing application.

A benefit of filtered scan is a reduced energy consumption. Indeed, an application that is not using a filter will get notified for every detection of BLE device within range, consequently the calling application will be woken-up more often and will be draining

*This research has been prepared before joining the CNIL. The views expressed do not necessarily reflect the views of the Commission or any individual Commissioner.

more battery. For the same reason, since Android 8, filtered scans are the only scan that can be run while the screen is off [16].

2.1.2 Batch scans. Instead of reporting each new scan result (i.e., each new detected device), it is possible to report the results by batch at a time specific interval [7]. To use batch scanning, an application specifies a positive scan delay when starting the scan and use the `onBatchScanResults()` method instead of the `onScanResult()` to retrieve the results. The `onBatchScanResults()` method is also part of the `ScanCallback()` object and is called whenever a batch of scan results is delivered.

Using batch scan is another mean to reduce energy consumption of applications performing BLE scans. Indeed, with batch scanning, the application will be notified and woken-up less often than with standard scanning.

2.2 Location on Android

On Android, an application can obtain the location of the device through the system's Location Manager API. In addition to GPS, Android can derive the location of the device from other sources such as cellular network and wireless network. For instance based on the list of nearby Wi-Fi and Bluetooth/BLE devices, the system can derive a location with an accuracy as low as 10 meters.

2.2.1 Location permissions. Being a sensitive information, the location is protected by a number of elements in the Android system. The first protection is implemented through the location permissions: `ACCESS_COARSE_LOCATION` and `ACCESS_FINE_LOCATION`. These permissions needs to be declared in the application manifest to allow an application to get access to location. These two permissions are enough to access the location while the application is running in foreground. To access location while in background, the `ACCESS_BACKGROUND_LOCATION` is required.

In addition to the declaration in the application manifest, permission to access location need to be granted by the user at runtime. While running, an application having declared one of the location permissions can request access to location. This access authorization can be granted permanently or on a one-time basis.

2.2.2 Location setting. Another element controlling the access to location information is the location setting, which defines whether the location service can provide a location information. Therefore, application (having the location permission) can only obtain location when this service is active.

If an application requests the access to location while the location setting is off, the user will be asked if it want to activate location.

When the location is enabled, an icon appears to inform the user that the location of the device is currently available, and that some application may use it. Indeed, once the location service is active, any application that has been granted one of the location permission can access the location information.

Furthermore, this location information can be used by the system and, unless the user disabled Google Location Services, is reported to Google. The Google Location Services is the location provider activated by default of most Android devices. According to Google documentation, when a device uses the Google Location Services, Google may collect location data periodically and use this data in an

anonymous way to improve location accuracy and location-based services.

3 ANDROID RESTRICTIONS ON BLE SCANNING

As result of BLE scans could be leveraged to derive geolocation, restrictions have been added to Android to protect abuse of the BLE features. Those restrictions apply to two main BLE features, namely start scanning and getting scan results, and are based on two elements: the location permissions and the status of the location setting. A summary of those restrictions is presented in Table 1.

3.1 Start scan & location setting

Starting Android 9, a constraint was added in the `ScanManager` class: to start a scan, an application must either use a filter or the Android location must be enabled [5]. This constraint is enforced in the `handleStartScan()` that is interrupted if the aforementioned conditions are not satisfied (see code 1)

```
329 final boolean locationEnabled = mLocationManager.  
    isLocationEnabled();  
330 if (!locationEnabled && !isFiltered) {  
331     Log.i(TAG, "Cannot start unfiltered scan in location-  
        off. This scan will be" + " resumed when location is  
        on: " + client.scannerId);  
332     mSuspendedScanClients.add(client);  
333     if (client.stats != null) {  
334         client.stats.recordScanSuspend(client.scannerId);  
335     }  
336     return;  
337 }
```

Listing 1: Extract of the `handleStartScan()` method of the `ScanManager` class that checks if location is enabled or if the scan is filtered.

To the best of our knowledge, this constraint has not been documented and the code update does not provide much details [15]. This constraint aims to verify that either the calling application is running a filtered scan or that the Android location setting is enabled. Note that this test done in the `ScanManager` class just before starting the scan. It does not verify that the calling function has the location permission.

3.2 Scan result & location permission

As early as 2015, Google added a protection to Android 6 [12]: location permissions are now required to obtain scan results [2]. As stated in the Android documentation [3] of the `StartScan()` method which an application calls to initiate a BLE scan: *"An app must hold `ACCESS_COARSE_LOCATION` or `ACCESS_FINE_LOCATION` permission in order to get results."* In later version of Android (9 and above), `ACCESS_FINE_LOCATION` was required.

This location permission constraint is implemented by the `hasScanResultPermission()` method (see code 2) which checks that an application has the appropriate permission before calling its callback to give it the scan result.

```
1089 /** Determines if the given scan client has the  
        appropriate permissions to receive callbacks. */  
1090 private boolean hasScanResultPermission(final ScanClient  
        client) {  
1091     final boolean requiresLocationEnabled =
```

No need to ask the Android: BLE scanning without the location permission

```
1092         getResources().getBoolean(R.bool.  
            strict_location_check);  
1093         final boolean locationEnabledSetting =  
1094             Settings.Secure.getInt(getContentResolver(),  
            Settings.Secure.LOCATION_MODE,  
1095             Settings.Secure.LOCATION_MODE_OFF) != Settings.  
            Secure.LOCATION_MODE_OFF;  
1096         final boolean locationEnabled =  
1097             !requiresLocationEnabled ||  
            locationEnabledSetting || client.legacyForegroundApp  
            ;  
1098         return (client.hasPeersMacAddressPermission || (  
            client.hasLocationPermission  
1099             && locationEnabled));  
1100     }
```

Listing 2: Code of the hasScanResultPermission in Android PIE [6]

The hasScanResultPermission() method is called in the onScanResult() method to determine whether a scan result should be reported to the application. Consequently, as suggested by the documentation [3], even if it could start a BLE scan an application without the location permission should always receive an empty list of results from the onScanResult() method [4].

Note that the other method able to receive scan results, onBatchScanResult(), does not perform a permission check via the hasScanResultPermission(). In fact, no verification regarding the location permission is implemented in the onBatchScanResult() method.

3.3 Scan result & location setting

Google has imposed a second constraint on Bluetooth: *"with privacy in mind, [Google] designed the Android operating system to prevent Bluetooth scanning unless the device location setting is on"* [24]. This constraint was also added in Android 6 [13] and is enforced, as the location permission constraint, by the hasScanResultPermission method (see code 2). If the location the setting is not enabled the application will receive an empty scan result.

On Android 8 and 9, a parameter called *strict_location_check* defined in a configuration file editable by device manufacturers could be edited to not require the location to be enabled for BLE scan to return results. Initially, this possibility was added for devices that had no location services (e.g., wearables) [14], but it seems that it was also used on smartphones [11, 28].

	filter	location permission	location enabled
startScan()	unfiltered		✓
	filtered		
onScanResult()	unfiltered	✓	✓
	filtered	✓	✓
onBatchScanResult()	unfiltered		
	filtered		

Table 1: Summary of the restriction on the BLE scanning API (a ✓ indicates that the item is required).

4 FLAWS IN ANDROID RESTRICTIONS

In this section, we detail two flaws in the restriction of Android’s BLE features. More specifically, we found how an application can start a BLE scan and get results without having the location permission. This flaw can be misused by Android application to determine the user location without his consent thus completely bypassing the restrictions implemented in Android since 2015. Furthermore, we discover a second flaw affecting scan filters that allows application to bypass the requirement on active location.

4.1 Bypassing the location requirements using batch scans

As presented in section 3, use of BLE scanning features is restricted by the location permission as it could be misused to locate unsuspecting users. It appears that the constraints preventing applications from misusing BLE are not coherently enforced and that an application can bypass both the location permission and the location setting requirements.

Enforcement of the constraints is distributed over different elements of the Android code. A first set of verifications is done when starting the scan but these verifications do not check whether the application has the location permission, and only checks that the location is enabled or that it is a filtered scan.

A second set of verification is done when scan results are collected through callbacks. Before returning the scan result, the systems checks that the application has the scan result permission using the hasScanResultPermission method. This verification is done in the onScanResult() method (see code 3), which handles the standard scan results. However, a similar verification cannot be found in the methods handling batch scan results: onBatchScanReports() and sendBatchScanResults()¹.

```
1003 // Do no report if location mode is OFF or the client has  
            no location permission  
1004 // PEERS_MAC_ADDRESS permission holders always get  
            results  
1005 if (!hasScanResultPermission(client) || !matchesFilters(  
            client, result)) {  
1006     continue;  
1007 }
```

Listing 3: Verification of scan result permission in onScanResult() in Android PIE [9]

Therefore, batch scan results are reported to the application without checking if it has the scan result permission. In other words, the application can collect batch scan results even if it does not have the location permission and even if the location is not enabled.

Regarding the location setting, we saw that it is required to start standard scans. Thus, an application will still need to activate the location to start the scanning, even if scan result could be obtained without it. We present a solution to remove this constraint in the next section.

¹This verification was added in the onBatchScanReports() method after we disclosed this vulnerability.

4.2 Abusing scan filters to scan while location is off

Filtered scans (see section 2.1.1) can be used to scan for specific devices matching a set of criteria. As seen in section 3.1, less constraints apply to filtered scans than to classic scans: filtered scans do not require the location to be enabled to be started, and they can run while the screen is off [17].

We found that, because of an implementation bug, it is possible to create a scan filter that will include all records. Such *pass-all* filter can be created by passing certain values to a method in charge of creating scan filters. More specifically, building a scan filter focused on the Manufacturer Specific Data while passing the null value for both the data and the datamask parameters will create a filter that do not exclude any result (see code 4).

```
1003 ScanFilter.Builder().setManufacturerData(42, null, null).
    build()
```

Listing 4: Creation of a bogus scan filter. The first parameters specifying a company ID is set to 42, while the data and datamask parameters are set to null.

The scan results obtained are identical to those of an unfiltered scan (all records are included), while the scan results will be considered as filtered. Because the scan is labelled as filtered, it can be started without the location to be active and can be performed while the application is in background.

This behavior could have been a simple bug, but because of the privileged handling of filtered scans, it allows application to scan all nearby devices without needing to activate the location setting.

4.2.1 Cause of the bug. This unintended behavior appears to be tied to the offloading of filtering on the hardware. Some Bluetooth controllers supports on-board filtering and filters can be passed by the OS through HCI commands² If filters are configured and if the hardware supports it, the filtering will be offloaded to the controller and the OS filtering emulation will be disabled (see code 5).

```
1003 BluetoothLeScannerCompat.java emulateFiltering = !
    filters.isEmpty() && (!offloadedFilteringSupported
    || !settings.getUseHardwareFilteringIfSupported());
```

Listing 5: Code of ScanCallbackWrapper() method disabling filtering emulation when hardware supports it (Android PIE)

When filtering is offloaded to the hardware, the OS will not apply the filters when receiving the scan results and will return them as is (see code 6).

```
1102 /* package */ void handleScanResults(@NonNull final List
    <ScanResult> results) {
1103     if (scanningStopped)
1104         return;
1105     List<ScanResult> filteredResults = results;
1106     if (emulateFiltering) {
1107         filteredResults = new ArrayList<>();
1108         for (final ScanResult result : results)
1109             if (matches(result))
1110                 filteredResults.add(result);
1111     }
1112     scanCallback.onBatchScanResults(filteredResults);
```

²https://source.android.com/devices/bluetooth/hci_requirements#advertising-packet-content-filter

```
1113 }
```

Listing 6: Code of handleScanResults() in which filters are not applied if the filtering emulation is not enabled

Therefore, the hardware is trusted with the filtering task and no verification is done by the OS on the returned scan results. We have been able to reproduce this behavior on all the devices we've have tested³.

We analyzed HCI communications between the Bluetooth controller and the host, and looked for the messages corresponding to the configuration of the filters. We found that when the filter is configured with non-null values for the data and datamask parameters, the expected command is transmitted by the host to the controller. However, when the data and datamask parameters are null, no command is transmitted and the filter is therefore not configured on the controller. This suggests that the problem is not coming from an implementation issue on the controller but rather from a problem on the host.

4.3 Summary of the flaws

The flaws we have identified are caused by the following behaviors:

- The location permission is not required to start batch scans nor to get batch scan results;
- The location setting is not required to get batch scans results;
- The location setting is not required to start filtered scans;
- On some devices, it is possible to run filtered scans that will keep all results.

Some of these flaws can be leveraged independently to bypass restrictions on BLE scans. Put together, they can be used fully circumvent the location restriction of the BLE scanning: collecting scan results of all nearby devices without requiring the location permission nor activation of the location setting. Such features would be obtained by an application that starts filtered batch scans with a pass-all filter and that collects scan results in batch.

4.4 Proof-of-concept

In order to demonstrate the vulnerability, we developed a proof-of-concept application. This application simply performs BLE batch scans (i.e. with a non-null reporting delay as explained in section 2.1.2) and displays detected devices. The application can run scans even when the application has not been granted the location permission, thus allowing to check that the constraint can be effectively bypassed. The application can run two types of scans:

- unfiltered scan that requires the android location to be enabled to obtain results.
- filtered scan that does not need the location to be enabled. By leveraging the scan filter bug, the scans include all records and the results are thus identical to the unfiltered one.

In addition to the code of the proof of concept, the application APK is also available online [10]. This proof of concept has been tested on devices running Android 8, 10 and 11. On devices running these three versions of the operating system, the proof of concept effectively displayed all nearby devices.

³Galaxy A5 (SM-A510F) Android 7, Galaxy A5 (SM-A520F) Android 8, Galaxy A40 (SM-A405FM) Android 10, Moto G5S Android 10 (LineageOS 17.1), Pixel 4a 5G Android 11

5 IMPACT ON MOBILE APPLICATIONS

5.1 Flaw used by malicious applications

It is possible that some applications already abuse those flaws to gain illegitimate access to BLE scan results. So far, we have not identified any application exploiting those flaws.

5.2 Scanning without permission

As developers have followed the Android documentation, they have taken into consideration the "location permission" constraint as it was clearly mentioned in Android documentation. Indeed, it appears that many applications performing BLE scan are actually requesting the location permission before starting BLE scan. Some applications performing batch scan (e.g. StopCovid) do require the location permission even though the permission is – as we have described in this paper – not enforced.

5.3 Scanning with location off

On the other hand, the undocumented restriction that requires location to be enabled in order to perform a scan has not always been handled by applications. Contact tracing applications like StopCovid and TraceTogether do not verify that the location is enabled before starting a BLE scan. This lack of test is inconsequential on Android devices that can perform batch scan because in these instances the application callbacks are not affected by the location constraint, but these applications may not perform effectively on the few Android devices [30] that do not support batch scanning.

5.4 The case of contact tracing applications

Many contact tracing applications use BLE scanning to detect proximity with other devices and eventually notify them if the device owner is positive to Covid-19. Most BLE based contact tracing applications use the Exposure Notification API but a few make direct use of Android BLE APIs.

Applications that use Exposure Notification APIs have to ask that the user turns-on the Android location but do not require the location permission as the Google Play Services, on which the Exposure Notification is built, already have the location permission. Unlike applications based on Exposure Notification, StopCovid [21] and TraceTogether [20] require the location permission – at least according to Android documentation – but neither of these applications condition their use to the activation of the Android location.

Due to the flaws discussed in this paper, on devices supporting batch scanning, none of the applications mentioned above must actually obtain the location permission or require that the device location is enabled. Indeed, these applications use filtered scan (as specific services IDs have been defined) and can use batch scanning instead of regular scan (as StopCovid already does).

6 FIXING THE FLAWS

The flaw allowing to bypass location permission has been disclosed to the Android security team and was addressed in the February 2021 Android security bulletin (CVE-2021-0328) [18].

6.1 Patching Android code

The location permission flaw has been fixed in Android code by calling the *hasScanResultPermission* from the callback function of *batchScan* to verify that, similarly to regular scans, the batch scan had the required permission to process the results [19]. This will prevent applications performing batch scanning from bypassing the location restrictions and would force them to obtain the location permission and to enable the location setting. While being convenient, this solution would address the problem on devices running up to date versions of Android however devices that no longer receive updates will remain indefinitely vulnerable.

The scan filter issue is likely the result of an implementation mistake and should be easy to fix. However, to avoid similar future issues, filters could be applied by OS even if the results returned by the controller are supposed to be already filtered.

6.2 App store enforcement

Applications abusing the location permission flaw could also be removed from the applications stores or blocked by device protection systems (e.g., Google Play Protect). Such enforcement could be performed by verifying that applications calling the *startScan()* method also have the location permission.

7 RELATED WORKS

Several works have demonstrated weaknesses in Android that would allow an application to obtain personal information. Some of those works focus on the abuse of wireless features to infer information as demonstrated in [23] where Wi-Fi scans can allow an application to obtain location without having the location permission. Focusing on Bluetooth, it has been demonstrated that identifiers associated to Bluetooth features can be used to perform cross-application tracking [26]. A large-scale collection of BLE scan results has been identified in [25]. Public information available in Android can be exploited to collect personal data [31]. Overall, those attacks leverage side channels to gain access to information otherwise protected by permissions [27]. Our work shows that, even if those threats have been identified for some time, protection mechanisms are still affected by serious flaws.

Illegitimate access to personal information can be caused by inconsistencies in the enforcement of restrictions [22, 29]. In [22] the author explains the causes of those inconsistencies and propose *AceDroid* a framework to identify them. Our work provides an additional illustration of these inconsistencies.

As noted in [32] and [22], some of those issues are caused by vendor customization of the OS. The *strict_location_check* setting discussed in Section 3.1 provides a typical example of how vendor can lower protection through customization.

8 CONCLUSION

This paper presented issues with the enforcement of location restrictions applying to Android's BLE scanning feature. The main issue is a vulnerability that allows application to bypass location permission associated with BLE scanning. This vulnerability has been assigned the identifier CVE-2021-0328. We identified a secondary issue that allows to bypass the need for the location to be active when scanning. By combining those flaws, an application

can fully bypass the location restrictions of BLE scanning and could thus be exploited to surreptitiously collect information (including location) on users. The main vulnerability associated to the CVE-2021-0328 has been fixed in Android but the patch will not reach devices that are not maintained, leaving millions (billions) users exposed.

These findings highlight the difficulty of implementing restriction in systems. In particular, when verification are spread over several location in the code, inconsistency in the checks is likely to appear [22]. Furthermore, we note that there are several inconsistencies between code and documentation as some restriction found in the code are not documented, and documented restrictions are not fully enforced.

ACKNOWLEDGMENTS

We would like to thank Monir Azraoui, Régis Chatellier, Jérôme Gorin and Vincent Rasneur for fruitful discussions. This research has been supported by the ANR-BMBF PIVOT and ANR DAPCODS projects.

REFERENCES

- [1] [n.d.]. Android developers documentation: Bluetooth Overview. <https://developer.android.com/guide/topics/connectivity/bluetooth#Permissions>. Accessed: 2021-03-25.
- [2] [n.d.]. Android documentation Android 6.0 Changes. <https://developer.android.com/about/versions/marshmallow/android-6.0-changes#behavior-hardware-id>. Accessed: 2021-03-25.
- [3] [n.d.]. Android documentation BluetoothLeScanner. [https://developer.android.com/reference/android/bluetooth/le/BluetoothLeScanner#startScan\(android.bluetooth.le.ScanCallback\)](https://developer.android.com/reference/android/bluetooth/le/BluetoothLeScanner#startScan(android.bluetooth.le.ScanCallback)). Accessed: 2021-03-25.
- [4] [n.d.]. Android Oreo source code GattService class. <https://android.googlesource.com/platform/packages/apps/Bluetooth/+/refs/heads/oreo-r6-release/src/com/android/bluetooth/gatt/GattService.java#859>. Accessed: 2021-03-25.
- [5] [n.d.]. Android source code ScanManager class. <https://android.googlesource.com/platform/packages/apps/Bluetooth/+/master/src/com/android/bluetooth/gatt/ScanManager.java#329>. Accessed: 2021-03-25.
- [6] [n.d.]. HandleScanResults. <https://android.googlesource.com/platform/packages/apps/Bluetooth/+/refs/heads/pie-security-release/src/com/android/bluetooth/gatt/GattService.java#1089>. Accessed: 2021-03-20.
- [7] [n.d.]. HCI Requirements. https://source.android.com/devices/bluetooth/hci_requirements#batching-of-scan-results. Accessed: 2021-03-25.
- [8] [n.d.]. iBeacon - Apple Developer. <https://developer.apple.com/ibeacon/>. Accessed: 2021-03-25.
- [9] [n.d.]. onScanResult. <https://android.googlesource.com/platform/packages/apps/Bluetooth/+/refs/heads/pie-security-release/src/com/android/bluetooth/gatt/GattService.java#1003>. Accessed: 2021-03-20.
- [10] [n.d.]. Proof of Concept : Android-Check-Location-required. <https://github.com/vtoubiana/Android-Check-Location-required>. Accessed: 2021-03-25.
- [11] [n.d.]. Xiaomi Mido Config file. https://git.aicp-rom.com/device_xiaomi_mido.git/tree/overlay/packages/apps/Bluetooth/res/values/config.xml?h=31f220353508077236e84d7655dd8fd90914060e. Accessed: 2021-03-25.
- [12] 2015. Android code commit: location permission constraint. https://cs.android.com/android/_/android/platform/packages/apps/Bluetooth/+/d2fc8cbd87c7a74223e8742a442a48690d426ce. Accessed: 2021-03-25.
- [13] 2015. Android code commit: location settings constraint. https://cs.android.com/android/_/android/platform/packages/apps/Bluetooth/+/312e10ad5bcb1e3d021c4798d55c40f99c7a6ef. Accessed: 2021-03-25.
- [14] 2015. Android code commit: make location check configurable. https://cs.android.com/android/_/android/platform/packages/apps/Bluetooth/+/72ecb4caa630b63f66505ecb202a807b1af4e294. Accessed: 2021-03-25.
- [15] 2015. Android code commit: ScanManager verifications. https://cs.android.com/android/_/android/platform/packages/apps/Bluetooth/+/efc013507b0b2b5c772f6db0389b5bd29116a0b7. Accessed: 2021-03-25.
- [16] 2017. Android 8.1 App compatibility w.r.t BLE Behaviour. <https://issuetracker.google.com/issues/70619940>. Accessed: 2021-03-25.
- [17] 2017. Stop unfiltered BLE scanning on screen off. https://cs.android.com/android/_/android/platform/packages/apps/Bluetooth/+/319aeae6f4ebd13678b4f77375d1804978c4a1e1. Accessed: 2021-03-25.
- [18] 2021. Android Security Bulletin—February 2021. <https://source.android.com/security/bulletin/2021-02-01>. Accessed: 2021-03-25.
- [19] 2021. Check permission before sending batch scan result. <https://android.googlesource.com/platform/packages/apps/Bluetooth/+/6f7f9bbf46acaaf266537256da4d0345909ea1c4%5E%21/#F0>.
- [20] 2021. OpenTrace Android App Source Code. <https://github.com/opentrace-community/opentrace-android>. Accessed: 2021-03-24.
- [21] 2021. StopCovid Source Code. <https://gitlab.inria.fr/stopcovid19/stopcovid-android>. Accessed: 2021-03-24.
- [22] Yousra Aafer, Jianjun Huang, Yi Sun, Xiangyu Zhang, Ninghui Li, and Chen Tian. 2018. AceDroid: Normalizing Diverse Android Access Control Checks for Inconsistency Detection. In *Proceedings 2018 Network and Distributed System Security Symposium*. Internet Society, San Diego, CA. <https://doi.org/10.14722/ndss.2018.23121>
- [23] Jagdish Prasad Acharya, Mathieu Cunche, Vincent Roca, and Aurélien Francillon. 2014. Short Paper: WifiLeaks: Underestimated Privacy Implications of the ACCESS_WIFI_STATE Android Permission. In *Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless & Mobile Networks* (Oxford, United Kingdom) (WiSec '14). ACM, 231–236. <https://doi.org/10.1145/2627393.2627399>
- [24] Dave Burke. 2020. An update on Exposure Notifications. <https://blog.google/inside-google/company-announcements/update-exposure-notifications>. Accessed: 2021-03-25.
- [25] Paul-Olivier Dehaye and Joel Reardon. 2020. Proximity Tracing in an Ecosystem of Surveillance Capitalism. *Proceedings of the 19th Workshop on Privacy in the Electronic Society* (Nov 2020). <https://doi.org/10.1145/3411497.3420219>
- [26] Aleksandra Korolova and Vinod Sharma. 2018. Cross-App Tracking via Nearby Bluetooth Low Energy Devices. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy* (Tempe, AZ, USA) (CODASPY '18). Association for Computing Machinery, New York, NY, USA, 43–52. <https://doi.org/10.1145/3176258.3176313>
- [27] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 2019. 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 603–620.
- [28] Darek Seweryn. 2019. A Curious Relationship: Android BLE and Location (via Webarchive). <https://web.archive.org/web/20201212091443/https://www.polidea.com/blog/a-curious-relationship-android-ble-and-location/>. Accessed: 2021-03-25.
- [29] Yuru Shao, Jason Ott, Qi Alfred Chen, Zhiyun Qian, and Z. Morley Mao. 2016. Kratos: Discovering Inconsistent Security Policy Enforcement in the Android Framework. In *Proceedings 2016 Network and Distributed System Security Symposium*. Internet Society, San Diego, CA. <https://doi.org/10.14722/ndss.2016.23046>
- [30] Martijn van Welie. 2019. Making Android BLE work — part 1. <https://medium.com/@martijn.van.welie/making-android-ble-work-part-1-a736dcd53b02>. Accessed: 2021-03-25.
- [31] Xiaoyong Zhou, Soteris Demetriou, Dongjing He, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, Carl A. Gunter, and Klara Nahrstedt. 2013. Identity, Location, Disease and More: Inferring Your Secrets from Android Public Resources. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS '13)*. ACM, New York, NY, USA, 1017–1028. <https://doi.org/10.1145/2508859.2516661>
- [32] Xiaoyong Zhou, Yeonjoon Lee, Nan Zhang, Muhammad Naveed, and XiaoFeng Wang. 2014. The peril of fragmentation: Security hazards in android device driver customizations. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 409–423.