Cross-App Tracking via Nearby Bluetooth Devices

Aleksandra Korolova
University of Southern California
korolova@usc.edu

Vinod Sharma
University of Southern California
vinodsha@usc.edu

I. Abstract

16 out of the top 100 free apps in the Google Play store "access Bluetooth settings" or "pair with Bluetooth devices." Although the promise of innovations that can be enabled by Bluetooth Low Energy technology is alluring, the privacy implications of it are poorly understood. This paper demonstrates that the Bluetooth Low Energy (BLE) protocol together with the Bluetooth permission model implemented in the Android and iOS operating systems can be used for device tracking unbeknownst to the individuals.

Specifically, through a series of experiments and analyses based on real-world smartphone data we show that:

- by listening to advertising packets broadcasted by nearby BLE-enabled devices and recording information contained in them, app developers can derive a fairly unique "fingerprint" for each of their users that can be subsequently used for cross-app tracking;
- the privacy protections put in place by the Bluetooth Special Interest Group, Google, and Apple are not sufficient to prevent such fingerprinting or to make cross-app tracking difficult to execute.

We propose mitigation strategies to decrease the feasibility of tracking using nearby BLE devices while preserving the utility of the BLE technology.

II. Introduction

Online tracking and profiling is one of the main privacy concerns of individuals today. The mechanisms of tracking when individuals access the Internet on their computers are well-understood by privacy researchers, legal scholars, and individuals themselves, and there have been significant efforts in many communities to empower individuals to be able to limit or make choices about tracking. In the technological tools space that includes browser privacy settings to control and remove cookies, extensions that block trackers, privacy settings to control third-party cookies, incognito mode, and so on; in the research space – investigation of browser, canvas, and other types of fingerprinting and remediations against them; in the policy space – the Do Not Track initiative, the Network Advertising Initiative[1], and others.

However, with the recent shift to smartphones and the rise of mobile applications as the primary interface to the Internet and between individuals and businesses, the question of what is being done to track individuals and what can be done to restrict it or give individuals choices regarding it arises anew. In a sense, mobile tracking and its circumvention is a new arms-race between mobile app developers and individuals. It is a high-stakes race for the former, because profile-based advertising is one of the main vehicles for online monetization and so better tracking may lead to better monetization; for the latter, because activity performed on smartphones has increasingly far-reaching privacy implications as the smartphone becomes the main device with which one interacts with the world.

From the app-developer perspective, cookies, the most prevalent tracking mechanism for desktops, are less useful in the mobile context as typically each mobile application runs in a separate sandbox that cannot share information with others. Thus, app developers interested in exchanging information about their users are constantly looking for new tracking approaches. The main tracking strategies rely on the following mechanisms:[2]

1) Device-specific identifiers, such as Apple's Unique Device ID, Google's Android ID, MAC address, etc.
2) Log-in credentials provided to the app, such as an email address, a phone number, or identity obtained through a set of authentication APIs (e.g., Facebook, OpenID, Google) provided by the user in order to sign up for the app.
3) Fingerprints of the user device derived from its properties using statistical techniques.

Concurrently to the development of techniques for smartphone-specific user tracking by app developers, mobile operating system developers Apple and Google and individuals have been looking for techniques to curb or circumvent tracking. For example, to address 1), Apple has moved away from making the Unique Device ID accessible to apps through the developer API[3], replacing it instead with an Advertising Identifier that can be reset by the phone owner at any time[4]. To address 2), individuals often choose to provide different credentials to different apps, and numerous services have arisen to aid in the generation of one-time email addresses or phone numbers that can be used for this purpose (e.g., https://throttlehq.com, http://www.burnerapp.com). Finally, to address 3), privacy researchers investigate techniques that may be used for fingerprinting phones and develop countermeasures [1], [2], [3], [4], [5], [6].

Our work brings to light a novel tracking technique of the third type, and, by creating awareness of its feasibility, argues for modifications needed to protect privacy. Specifically, we observe that developer APIs on both Android and iOS

[1] https://www.networkadvertising.org/about-nai

[2] http://www.allaboutcookies.org/mobile/mobile-tracking.html

[3] http://now.avg.com/apple-ios-7-puts-unique-device-ids

[4] https://support.apple.com/en-us/HT205223

do not require a user's permission for an app to run a scan that listens for presence announcements from nearby Bluetooth Low Energy (BLE) devices[5]. We hypothesize that, even though the Bluetooth Special Interest Group, Apple, and Google have taken some steps to decrease the potential privacy impact of such scans, those steps are not sufficient to prevent cross-app tracking. Specifically, we hypothesize that by running periodic scans for nearby BLE devices and recording the information contained in them, app developers could derive a fairly unique "fingerprint" for each of their users. Use of these fingerprints could power a new type of tracking, that would enable app developers to identify users across apps and across devices without reliance on shared log-in credentials or other device identifiers. We hypothesize that derivation of a fairly unique fingerprint from observations of packets from nearby BLE peripherals is feasible due to the rapidly increasing number and diversity of BLE devices, virtually unfettered access to BLE data in Android and iOS, and information-theoretic richness of that data.

To verify our hypothesis of the feasibility of cross-app tracking via nearby BLE devices, we build Android and iOS apps that run scans listening for presence announcements of BLE devices with fixed periodicity and record the results. We discover that due to Apple's design choices related to the Bluetooth API implementation, cross-app tracking is trivial for iPhone developers. We then incentivize 100 individuals from a university to install and keep our Android app running on their mobile devices for a week. We analyze the data obtained and find strong support for our hypothesis also on Android; namely, that the data of each scan is rich enough to make it feasible for multiple app developers to identify their shared users even in the absence of other identifiers.

Cross app-tracking using BLE data constitutes a serious privacy threat as under the current implementations of BLE protocol and mobile APIs and permission models it could happen without the individual's knowledge or consent, and with limited ability for them to detect or prevent it. Although one may argue that app developers can use advertising identifiers to track users, and thus there is hardly a need to worry about possible more sophisticated tracking, such as BLE tracking, that is not the case, at least conceptually, from the users' standpoint. The advertising identifier can be reset in Android and iOS, and its tracking implications are well-publicized and well-understood. Thus the privacy-conscious users have some recourse to limit advertising id-based tracking. Furthermore, removal of an advertising identifier altogether by Apple and Google, or automated frequent resets of it, is merely a matter of policy, not technology. Currently, there is no such recourse for BLE-based tracking, for two reasons. First, the feasibility of such tracking prior to our work was unknown and, second, the mobile ecosystem does not provide tools to curb it (we consider the idea of keeping the phone's Bluetooth off at all times incompatible with functionality). We hope that as a result of this work, the privacy implications of BLE technology for tracking will become better understood and that the Bluetooth Special Interest Group, Bluetooth device manufacturers, and mobile OS developers Apple and Google will implement changes that will give individuals ability to limit such tracking if they so choose, without having to give up on using innovative technologies enabled by BLE.

The rest of the paper is organized as follows. Section III provides background on the BLE protocol and related privacy features put forth by the Bluetooth Consortium, Apple, and Google. Section IV describes the Android app we built, the design of our data collection study, and relevant characteristics of the obtained dataset. Section V presents the data analyses and experiments we ran to support our hypothesis of feasibility of cross-app tracking using nearby BLE devices on Android. Section VI explains why BLE-based tracking is trivial on iOS. We discuss the implications of our findings, including possible approaches to decreasing the feasibility of cross-app tracking using nearby BLE devices in Section VII. Finally, Section VIII describes related work.

III. BLE Background & Privacy Features

BLE (aka LE) is a Bluetooth protocol introduced in June 2010 by the Bluetooth Special Interest Group designed to enable a new set of devices with low power consumption [7][6]. Under this protocol, each BLE *peripheral* device, such as a fitness tracker, announces its presence to all nearby *central* devices, such as phones or computers, through advertising. Advertising consists of 8-39 byte advertising packets broadcasted by BLE peripherals through three dedicated advertising channels in the 2.4GHz ISM band. The advertising packets are sent with a periodicity of 20ms to 10.24 seconds.[7] The distance range within which the advertising packets may be read varies depending on the transmit power of the peripheral device, and can be as large as 100 meters.[8]

We refer to a central device that merely listens to the dedicated advertising channels and thus discovers nearby BLE devices as a *passive scanner*. In addition to learning information by listening for advertisements, a central device can also send *scan requests* and *connection requests* to peripherals. In this work, we focus on cross-app tracking that can be performed by passive scanning alone, and therefore, cannot be detected by peripherals as it does not communicate any information to peripherals. Subsequently, we refer to phones acting as central devices as scanners.

A. Data Available in Passive Scanning Phase

We now describe the information typically contained in advertising packets sent by peripherals. The contents of the advertising packet can vary, depending on the peripheral, but each advertisement always includes a 2-byte header and a 6-byte advertisement *address*, typically referred to as the Bluetooth MAC address or simply, the address of the peripheral, and, optionally, up to 31 bytes of other data[9]. Besides the peripheral address, some of the data types frequently included that may be useful for cross-app tracking are:

- *Service UUID*: a unique identifier for each service provided by the peripheral. Each advertisement contains

[5]Starting with Android 6.0, a location permission is required, as discussed in Section III-C.

[6]https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics/low-energy

[7]http://www.argenox.com/a-ble-advertising-primer

[8]https://www.sans.edu/research/security-laboratory/article/bluetooth

[9]https://www.bluetooth.org/en-us/specification/assigned-numbers/generic-access-profile

zero or multiple service UUIDs and the same service UUID can be advertised by different peripherals. For manufacturer-specific service UUIDs, only different peripherals which are of similar type (e.g., all fitness trackers of the same brand and type) will advertise the same service UUID.

- *Peripheral Name*: a name of the peripheral device. Different peripherals can have the same name if their manufacturer has assigned a fixed name to its devices.
- *Transmitting Power Level*: the current (numerical) transmitting power level of the peripheral device. Although this value can be changed by users, typically it remains unchanged from the value set by the manufacturer of the peripheral.
- *Manufacturer Data*: information provided by the specific manufacturer in binary format. This field contains two or more octets. The first two octets contain manufacturer IDs and the format of the other octets depends on the specific manufacturer and usually is not public knowledge. The manufacturer ID (aka company ID)[10] is a 16-bit number assigned to Bluetooth SIG members.
- *RSSI*: the current received signal strength indicator of the peripheral (in decibels). RSSI is a mandatory field and indicates the closeness of the peripheral to the scanner and changes continuously in every new advertisement.

B. *Peripheral Address Privacy*

The Bluetooth Special Interest Group recognized that much like MAC addresses transmitted in other contexts [8], Bluetooth peripheral addresses can pose a privacy risk[11]. Specifically, they observed that a person who carries a BLE device with them throughout the day can be tracked using the MAC address that peripheral broadcasts in its advertisements. To remedy this, they introduced an "LE Privacy" feature that enables the peripheral MAC address broadcast within the advertising packets to be replaced with a random value that changes at timing intervals chosen by the manufacturer of the peripheral device. However, according to a recent survey by [9], many manufacturers do not properly implement this feature. A notable exception is Apple who ensures all their devices change their Bluetooth address every 15 minutes.

1) Treatment of Peripheral Addresses by Android and iOS: Both Android and iOS implement APIs methods that enable app developers to obtain information contained in BLE advertising packets. The Android APIs provide app developers with the peripheral addresses as they are presented in advertising packets. In other words, all Android applications running on all phones receiving advertising packets from the same peripheral at the same time will obtain the same peripheral address.

Unlike on Android, applications installed on iOS are provided with a 128-bit number called `peripheral uuid` via the Core Bluetooth API[12] instead of the peripheral address. This is a privacy feature introduced by Apple as a countermeasure against tracking [10] – Apple replaces the BLE peripheral address with a randomized peripheral uuid. The randomization procedure for transforming peripheral addresses to reported uuids is not publicly known. However, through our experiments we observe the following pattern: if two apps are installed on the same phone, the peripheral uuids of a peripheral with address p that they receive through the API will be identical; if two apps are installed on different phones, the peripheral uuids of a peripheral with address p will be different. Table I schematically shows the values of the peripheral addresses reported through the APIs for a fixed peripheral with address p as observed by two different applications running on two iPhones and two Android Phones.

	iOS		Android	
Apps	phone1	phone2	phone3	phone4
app1	x	y	p	p
app2	x	y	p	p

TABLE I. ADDRESSES OBSERVED BY TWO IOS AND TWO ANDROID APPLICATIONS RUNNING ON FOUR DIFFERENT PHONES FOR A PERIPHERAL ADVERTISING ADDRESS p

We will refer to the peripheral addresses that apps obtain through Android and iOS APIs as peripheral uuids in subsequent discussions in order to reflect the fact that the mobile OSes have the liberty to change the peripheral addresses that they present to app developers through their APIs.

As will become clear from subsequent discussion in Section VI, the fact that Apple's randomization ensures that a peripheral uuid of a particular peripheral is seen as the same for apps installed on the same phone, but as different for apps installed on different phones, makes cross-app tracking using nearby BLE devices on iOS trivial.

C. *Application Permissions Needed to Run Scans*

We now describe what is needed for an application installed on an iOS or Android-based phone to passively scan for nearby BLE devices, that is, obtain the contents of advertising packets broadcast by nearby BLE devices through the APIs.

1) iOS: An iOS application does not require any permissions to run Bluetooth scans if the app is running in the foreground, allowing any application to conduct scans for nearby Bluetooth devices. Although an iOS application can also run scans in the background using bluetooth-central background mode provided by the iOS, scans in the background mode are limited, i.e., they can scan only for BLE devices supporting specific services[13] [14].

[10]https://www.bluetooth.com/specifications/assigned-numbers/company-Identifiers

[11]http://blog.bluetooth.com/bluetooth-technology-protecting-your-privacy/, https://developer.bluetooth.org/TechnologyOverview/pages/le-security.aspx

[12]https://developer.apple.com/library/mac/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/AboutCoreBluetooth/Introduction.html

[13]https://developer.apple.com/library/ios/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/CoreBluetoothBackgroundProcessingForIOSApps/PerformingTasksWhileYourAppIsInTheBackground.html

[14]https://developer.apple.com/library/ios/documentation/CoreBluetooth/Reference/CBCentralManager_Class/#//apple_ref/occ/instm/CBCentralManager/scanForPeripheralsWithServices:options:

2) Android: An Android application that wants to run scans requires the BLUETOOTH and BLUETOOTH_ADMIN permissions. However, these permissions are automatically granted to an app that declares them in their manifest, because they belong to the "normal" protection level. The Android divides system permissions into protection levels, the most common of which are *normal* and *dangerous*[15]. Normal permissions are those whose access, according to Android's documentation, poses "very little risk to the user's privacy", and thus are automatically granted.

Starting with Android 6.0 (API level 23), in addition to the BLUETOOTH and BLUETOOTH_ADMIN permissions, applications also need ACCESS_COARSE_LOCATION or ACCESS_FINE_LOCATION permissions to run BLE scans[16]. Both of these location-related permissions belong to the "dangerous" protection level and thus require an explicit approval from the user of the app.

However, a malicious application developer can overcome the need to obtain a user's explicit permission by exploiting forward compatibility[17] features of Android. A developer can set the application target integer specifying the desired API level, `targetSdkVersion`, to 22 or smaller and thus bypass the need to ask a user's approval for accessing location when the app desires to run BLE scans. As of November 2016, 76.7% of Android devices are running with API 22 or lower[18], which suggests that many apps could be willing to give up on the new features of API level 23 to preserve ability to perform surreptitious cross-app tracking.

IV. A Real-World Dataset of BLE Scan Data

A. Data Collection

In order to perform a realistic evaluation of the feasibility of cross-app tracking using nearby BLE devices, we developed an Android app that uses Android's Bluetooth API[19] to conduct scans while running in the background. Our app conducted scans every 10 minutes as follows:

- Start a scan, scan for 1 minute, stop the scan
- Wait for 1 minute
- Start a scan, scan for 1 minute, stop the scan

In the period between April 25, 2016, and May 6, 2016 through our university's mailing lists we recruited 100 volunteers with phones running Android 5.1 lollypop or higher to install our app and leave it running on their phone for a week. 46 individuals installed the app between Apr 25-27, 20 – between Apr 28 - May 3, 34 – between May 4-6. Of those 100 individuals, 70 left our app running on their phone for a full week, and are the ones whose data we will be considering in subsequent analysis.

[15]https://developer.android.com/guide/topics/security/permissions.html#normal-dangerous

[16]https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html

[17]https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#fc

[18]https://developer.android.com/about/dashboards/index.html

[19]https://developer.android.com/guide/topics/connectivity/bluetooth.html

1) Ethics: Our institution did not require IRB approval since we took the following measures to protect the privacy of our volunteers:

- No registration of any kind was required, and thus we did not collect names, emails, phone numbers, or any other identifying information.
- The app was distributed through a public web page and required no interaction between study participants and researchers. The modest compensation for participation – an electronic gift card to a party unrelated to the researchers – was distributed by displaying its code in the user interface of the app a week after the app's installation.
- We disclosed the purpose of the app and collected only information needed for that purpose (i.e., only information contained in the Bluetooth advertisements). In particular, we did not collect location data, which limited the analyses we could run on the data obtained.

B. Dataset Characteristics

In the subsequent text, we refer to the phones on which the participants installed the app as scanners.

Our collected data provides evidence that most individuals (at least among those on a college campus) are near a variety of BLE-enabled devices much of the time.

Given our app's scanning behavior, each scanner that used our app for a week conducted 2,016 scans (2 scans every 10 minutes for 7 days). We call a scan during which a scanner receives some BLE advertisements *non-empty*. Each scanner had some non-empty scans: 57 out of the 70 scanners observed their first peripheral within 10 minutes of installing our app, 65 – within 1 hour of installing our app. The median number of non-empty scans for a scanner is 870, and 54 out of 70 scanners have carried out at least 504 non-empty scans, meaning that for 54 scanners, at least $\frac{1}{4}$-th of the scans conducted were non-empty.

The scanners together observed 45,283 distinct peripheral uuids.[20] Each scanner observed as few as 7 and as many as 2,429 distinct peripheral uuids, with the median number of distinct peripheral uuids observed by a scanner of 658. Most of the scanners observe a high number of peripheral uuids that are unique to them.

47 scanners observed the same peripheral uuid in 30% or more of their non-empty scans, suggesting that most scanners consistently and frequently see the same peripheral, and the peripheral uuid of the most commonly seen peripheral may provide a "fingerprint" or unique identifier for the phone. Furthermore, most scanners see many peripheral uuids that are not seen by any other scanner in the dataset, suggesting that these peripherals can also meaningfully contribute to the fingerprint. Detailed information for each scanner is presented in Table II.

[20]Recall from our discussion in Section III-B that observing 45,283 distinct peripheral uuids does not imply that there were as many distinct bluetooth-enabled devices near our volunteers, as some peripherals (most notably, Apple's devices) implement the "LE privacy" feature of periodically changing the peripheral address advertised.

Scanner #	# of non-empty scans observed by scanner	# of distinct peripheral uuids observed by scanner	# of scans in which the most frequently seen peripheral uuid of this scanner is observed	# of peripheral uuids not observed by any other scanner	fraction of non-empty scans that contain the same peripheral
1	2011	273	2011	273	1
2	1926	1148	1026	963	0.53
3	1848	1360	1388	1285	0.75
4	1842	965	1323	781	0.72
5	1834	1424	803	1290	0.44
6	1749	1350	1218	856	0.7
7	1721	1703	684	1204	0.4
8	1674	1228	1305	1200	0.78
9	1650	897	1407	612	0.85
10	1622	2429	795	2429	0.49
11	1612	2007	668	1443	0.41
12	1554	2216	935	1778	0.6
13	1491	424	1181	343	0.79
14	1450	1613	980	1127	0.68
15	1443	1003	608	932	0.42
16	1441	755	1021	409	0.71
17	1437	862	412	605	0.29
18	1433	333	1346	205	0.94
19	1407	1345	387	1120	0.28
20	1188	339	840	233	0.71
21	1168	1510	537	985	0.46
22	1152	282	521	163	0.45
23	1149	1269	89	946	0.08
24	1132	538	292	509	0.26
25	1130	963	561	875	0.5
26	1116	860	714	776	0.64
27	1112	1388	565	1190	0.51
28	1081	576	773	305	0.72
29	1076	922	397	680	0.37
30	1067	1333	83	885	0.08
31	1000	676	277	443	0.28
32	948	900	160	738	0.17
33	918	290	578	165	0.63
34	918	585	227	399	0.25
35	873	733	108	557	0.12
36	867	415	393	377	0.45
37	832	2040	313	1430	0.38
38	817	891	801	564	0.98
39	817	907	70	595	0.09
40	800	652	281	589	0.35
41	796	1254	73	1010	0.09
42	790	1116	333	805	0.42
43	735	404	129	240	0.18
44	732	702	394	596	0.54
45	694	446	559	369	0.81
46	674	1020	172	491	0.26
47	668	940	127	667	0.19
48	661	284	214	206	0.32
49	620	561	124	377	0.2
50	578	326	108	220	0.19
51	576	334	356	222	0.62
52	549	457	224	393	0.41
53	524	654	199	449	0.38
54	519	308	172	273	0.33
55	475	272	304	228	0.64
56	443	180	427	142	0.96
57	396	513	38	333	0.1
58	385	160	8	160	0.02
59	378	462	108	413	0.29
60	336	327	125	181	0.37
61	329	46	253	33	0.77
62	276	661	152	540	0.55
63	254	202	33	158	0.13
64	156	58	32	41	0.21
65	119	153	36	121	0.3
66	116	48	36	43	0.31
67	69	106	36	17	0.52
68	27	61	5	35	0.19
69	25	69	6	36	0.24
70	17	7	9	6	0.53

TABLE II. STATISTICS, PER SCANNER, ON THE NUMBER OF NON-EMPTY SCANS, DISTINCT PERIPHERAL UUIDS, FREQUENCY OF THE MOST FREQUENT PERIPHERAL, NUMBER OF PERIPHERALS UNIQUE TO A SCANNER.

Finally, for each of the peripheral uuids in our dataset, we compute the number of different scanners that see this peripheral uuid. There are a number of peripheral uuids that are seen by many scanners; for example, one of the peripheral uuids in our dataset is seen by 36 out of the 70 scanners, another – by 33 scanners, two others – by 30 scanners, and so on. Detailed information on the number of peripherals that are observed by a fixed number of distinct scanners is presented in Table III.

There are a number of reasons why a peripheral uuid may be observed by many scanners: it could be a stationary BLE device, such as a beacon, that many individuals pass by, or it could be a BLE device of some individual who meets many of the other individuals in our dataset (e.g., during a crowded campus-wide event). The fact that our dataset contains some peripheral uuids that are observed by more than one scanner is a good sanity check – it shows that our users are not completely disconnected from each other, and at least occasionally, are near each other or visit the same places. The peripheral uuids that are seen by many scanners make the task of fingerprinting harder, but as we will see next, not insurmountable. Furthermore, the vast majority of peripheral uuids (41,064) are observed by at most one scanner.

N	Number of peripheral uuids that are observed by exactly N distinct scanners
1	41064
2	2756
3	696
4	307
5	147
6	95
7	54
8	41
9	33
10	15
11	17
12	13
13	2
14	9
15	9
17	4
18	2
19	4
20	2
21	3
22	3
24	1
25	1
28	1
30	2
33	1
36	1

TABLE III. CHARACTERIZING HOW MANY SCANNERS OBSERVE THE SAME PERIPHERAL UUDS

V. CROSS-APP TRACKING FEASIBILITY ON ANDROID

In this section, we present experiments based on the data collected aimed to illustrate that our hypothesis of the feasibility of cross-app tracking via nearby BLE devices holds. We do not utilize the full set of information available to app developers from BLE scans; rather, we demonstrate that the tracking is feasible even when the only information used are the peripheral uuids collected in those scans. Our goal is to present a proof-of-concept that uses only the most basic data and relies on the most basic algorithms. In practice, the cross-app tracking can be more successful than shown in our experiments, as app developers can build sophisticated algorithms that utilize the full set of data.

A. Experimental Set-Up

The problem of cross-app tracking can be formulated as a problem of finding matching users between two applications (App1 and App2) based on the BLE scan data each application possesses for each user. We assume that the apps are sharing the scan data with each other. For ease of exposition, when we refer to a "user", we mean the BLE scan data an app has collected for that user. Specifically, for experiments in this section, each user U of an app A is represented by a (numpy) array, whose size is equal to the total number of distinct peripheral uuids observed in our data and whose array entry j corresponds to the number of scans in which app A running on user U's phone observed peripheral uuid j (we create a 1-1 mapping between peripheral uuids and array indexes for simplicity).

We assume the collaborating apps deploy the simple matching strategy described in Algorithm 1.

Algorithm 1: Match App1 Users with App2 Users

```
1 For each user $U$ of App1 compute his similarity score
  with each user of App2.
2 Select the user of App2 with the highest similarity
  score to $U$ as the matching user.
```

For simplicity, we assume that Apps 1 and 2 are used by the same set of users.

It remains to detail how we transform our data collected from a single app into simulated data from multiple apps, and how we compute the similarity score between users and the accuracy of the matching, which we do next.

1) Modeling Data From Multiple Apps: Our matching experiment requires data from multiple apps, but as explained in Section IV, the scan data collected by our volunteers came from one app. We could have chosen to ask our volunteers to install multiple apps, but the increased effort needed could have deterred some volunteers from participating in our study. Instead, we chose to use the data collected by our app that does very frequent scans, in order to model data collected by multiple apps doing less frequent scans.

Specifically, we model scan data that would be obtained from two applications running on the phone of the same user by splitting the data collected by our app on each scanner into two scanner instances. Suppose we want to model that each app is used by a user every $10x$ minutes, where x is an integer from 1 to 144. Then we assign data from scans numbered $i \cdot 2x$, where i is an integer starting at 0 to App 1, and data from scans numbered $x + i \cdot 2x$ to App2 (scans are numbered in the increasing order of the time at which they are run). Given the frequency with which our Android app runs scans (Section IV) this corresponds to Apps 1 and 2 being used by the user within 1 minute of each other when $x = 1$, and within $5x$ minutes of each other for larger values of x. By varying x between 1 and 144, we can model apps that are used as frequently as every 10 minutes, and as infrequently as once a day.

A distinct advantage of using the modeling approach, rather than asking our users to install multiple apps, is that we can model apps doing scans with different intervals between them. Via modeling, we also automatically obtain the "ground truth" of who are the matching users between apps. A disadvantage is that the simulated data results in scans at fixed intervals, which is more structured than what application developers might see in practice[21].

2) Similarity Scores: Given two arrays representing the scan data of two users held by two different apps, we would like to estimate the likelihood that the scan data belongs to the same user. We do so by computing a similarity score between the arrays. We use the off-the-shelf *Cosine* similarity score.

Given two arrays x and y of size n, each containing the number of scans in which each peripheral uuid has occurred for that user, the *Cosine* similarity score are computed as follows:

If $(\forall i, x_i = 0) \vee (\forall j, y_j = 0)$ then $\textit{Cosine}(x, y) = 0$; otherwise $\textit{Cosine}(x, y) = \frac{\sum_{i=0}^{n-1} x_i y_i}{\left(\sum_{i=0}^{n-1} \sqrt{x_i^2}\right)\left(\sum_{i=0}^{n-1} \sqrt{y_i^2}\right)}$.

The intuition behind using cosine similarity as a similarity measure is as follows. Two applications can see the same peripheral uuid during scans for BLE devices if: they are installed on the same phone OR they are installed on different phones, but the owners of those phones have passed by the same peripheral (not necessarily at the same time). If one application sees a particular peripheral uuid during scans, and another does not, it could be because the peripheral changes its address frequently or because these applications are running on different phones. The more common peripheral uuids the apps see and the more often it happens, the more likely these apps are to run on the same phone; the more distinct peripheral uuids the apps see and the more often that happens, the less likely these apps are to run on the same phone. The cosine similarity measure is just one option to encoding this, Jaccard similarity and many others would also be suitable. We deliberately do not optimize the similarity measure in order to demonstrate that tracking is feasible even without any optimization.

B. Experiment Results Using Peripheral UUIDs

Our results unequivocally demonstrate that cross-app tracking using nearby BLE-devices is a realistic possibility. Modeling frequency of app usage from every 10 mins to once a day, we split our data into two apps with 70 different users each. For each user of App 1, we computed his cosine similarity score with each user of App2 and select the user of App2 with the highest similarity score as the candidate matching user as per Algorithm 1. We present the number of correct matches (out of 70 possible) made by this algorithm depending on the app usage frequency in Figure 1. Even when the apps are used only once a day, more than half the users are matched correctly. This is an impressive result, as the expected number of users matched correctly by a random matching is one.

VI. Cross-app Tracking Feasibility on iOS

We also investigated the possibility of cross-app tracking using nearby BLE devices on iOS, and concluded that cross-app tracking is even easier on iOS than on Android, due to the differences in how Apple and Android transform the peripheral

[21]This disadvantage can be mitigated via modeling that sub-samples scans using a randomized, rather than a deterministic, process.

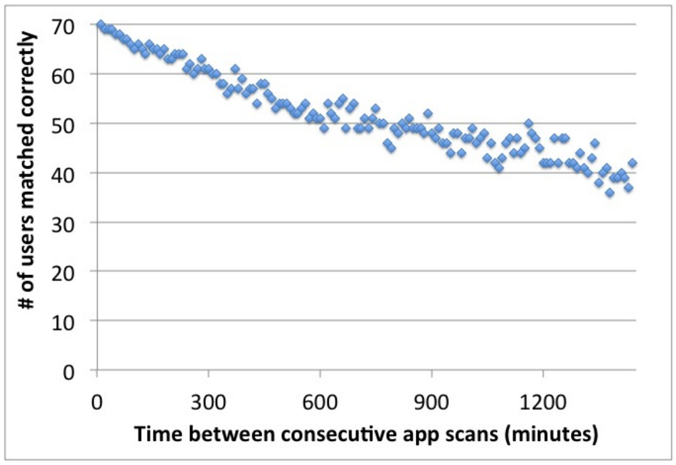

Fig. 1. Number of correctly matched users (out of 70) using cosine similarity score as a function of app usage frequency.

addresses received in advertisements before giving them to app developers through their APIs (Section III-B1 and Table I) and lack of permissions needed to run scans (Section III-C).

Since iOS limits the ability to run scans for all BLE devices when the application is in the background[22], it makes frequent data collection by volunteers more difficult, as they have to actively engage with our app. Thus, rather than trying to collect large-scale iOS data from volunteers, we focused on identifying possible differences between BLE data presented to app developers in iOS vs in Android. To that end, we developed an app for iOS identical to the one described in Section IV with the exception that an individual has to specify manually when the app should run a scan for BLE devices. In parallel, we modified our Android app to permit manual specification of when the scans should be run. For 4 days in May 2016, we manually ran two scans within one-minute interval of each other on an iPhone 5S running iOS 9.0 and on an Android device (Motorola Moto G3 running Android 6.0) at 18 distinct locations on our university's campus and recorded the data received. Our iOS application was compiled with *iOS Deployment Target* set to 9.0 and our Android application was compiled using *targetSDKVersion* set to 22. We observed no quantitative or qualitative differences in the data, except when scans were conducted at one particular location. In that location, our iOS app observed more distinct peripheral devices than our Android app, which can likely be attributed to a slight variation in the different vendor implementation of the BLE protocol for a particular BLE device located in that location.

We then ran two identical versions (differing only in the app's name) on two identical iPhones and two identical Android devices, performing scans both at various locations on campus, as well as in the authors' homes and offices, where the set of all BLE-enabled devices was known to us. It is through these experiments we discovered and verified the difference between Android's and iOS's Bluetooth APIs described in Section III-B1. Namely, a peripheral with address p will appear as having address p in (simultaneously run) scans of all apps installed on all Android phones, whereas the peripheral uuids seen by iPhone apps in advertisements coming from peripheral with address p will be different if these apps are installed on different iPhones but the same if they are installed on the same iPhone (see again Table I). This feature, introduced by Apple for privacy [10], has the (presumably unintended) consequence of making cross-app tracking using nearby BLE devices trivial on iOS. Indeed, if there is even one peripheral uuid that appears in the scan data of both applications, these applications can with certainty conclude that the scan data belongs to the same user. If apps installed on the same phone are conducting scans within a short timeframe of each other or if the user is near the same BLE peripheral during large chunks of the day (which our data of Section IV-B supports), the apps are virtually guaranteed to observe at least one common peripheral uuid.

[22]https://developer.apple.com/library/ios/documentation/CoreBluetooth/Reference/CBCentralManager_Class/#//apple_ref/occ/instm/CBCentralManager/scanForPeripheralsWithServices:options:

VII. Discussion

A. Study Limitations and Feasible Improvements

The experiments and analyses we presented have many caveats and limitations. For example, we conducted the study with volunteers who visit a university campus, where one can presumably find more BLE devices than in a poor residential neighborhood. Our data collection was done using one app, and then we sub-divided this data to simulate data from multiple apps running on the same phone, resulting in scans that were done with more regularity than one would expect from typical mobile app usage. When computing matching accuracy scores, we assumed the apps trying to match users had the same set of users. The list goes on.

These are all valid criticisms, some of which can be addressed with additional research. However, it was not our goal to provide a definitive robust cross-app tracking technology for today's BLE device landscape. Rather, our goal was to demonstrate that the current treatment of BLE, if unchanged, will give rise to possibilities of fairly successful cross-app user tracking in the future, and thus, to argue for modifications needed from Bluetooth Special Interest Group, Apple, Google, and BLE device manufacturers to safeguard user privacy before BLE-based tracking becomes widespread. Thus, we have deliberately used unsophisticated techniques to demonstrate the feasibility of cross-app tracking using nearby BLE devices. We could have increased the accuracy of our tracking by using more sophisticated approaches such as:

- relying on a more complex similarity score, including one derived using machine learning on part of the data (in practice, apps may know which of their users match through other means for a fraction of the users, and can use that data for training),
- distinguishing between advertisements received at nights vs during the day, to take advantage of the knowledge that users tend to be at home at night,
- building a more complex feature vector, that takes into account all information contained in advertisements (Section III-A), including Service UUIDs, manufacturer data, the time when the advertisement is received, the RSSI, Transmitting Power Level, peripheral name and others, periodicity with which advertisements are received from a peripheral with a particular uuid, etc.,

- paying particular attention to paired BLE devices, rather than relying on passive scanning,[23]
- obtaining information about manufacturer specific settings for common BLE devices and incorporating it into our feature vector building and similarity scoring functions.

Furthermore, with time, the number and variety of BLE devices used will increase, and so will the richness of the data, thus making cross-app tracking easier.

Our study was done using data of only 70 users who are all in some way connected to our university. A natural question is what happens when the number of users applications have is orders of magnitude bigger. We are interested in answering this question ourselves but a significantly more extensive study is not feasible in an academic setting. However, even though our dataset is small, it has characteristics that make cross-app tracking in it more difficult than for an arbitrary set of users. Specifically, because our volunteers live or work or study in close proximity, observations of identical peripheral uuids from different phones are more likely than they would be for a population of users that never overlaps in space (see Table III).

Finally, even if BLE-based tracking alone is not sufficient for full accuracy, our findings show that the data carries enough signal that it could amplify other forms of tracking. Thus, despite the limitations of our study, we believe that given the gravity of the privacy risk, the experiments presented are sufficient to confirm the viability of cross-app tracking and motivate a call for privacy-enhancing changes.

Cross-app tracking using nearby BLE devices is indeed a serious privacy risk as this tracking happens without an individual's knowledge. Furthermore, even a knowledgeable individual has no recourse other than to completely disable Bluetooth functionality on their phone, as there are no equivalents to "clearing cookies" or "resetting an advertising identifier" in the BLE tracking context. What could be learned about an individual in the course of such tracking is virtually unlimited – imagine an insurance app, a medical app, and a financial app collaborating to cross-app track users, and once the match is made, sharing information learned from within-app activities among each other.

B. *Building Richer Profiles Using nearby BLE Devices*

Although it was not a focus of our study, we observe that information collected via nearby BLE devices can also be used to refine a profile an app may have about the user. Specifically, our dataset contains more than 1,000 distinct peripheral names, some of which could be quite useful for profiling. For example,

- an app that observes a BLE device with name "Alice Smith's Fitbit" in most of its scans can conclude that the user's name is likely Alice Smith, a female,
- an app that often observes a BLE device with name "mamaRoo" in its evening and night scans can conclude that the user lives in a household with an infant,
- an app that observes a BLE device with name "[TV] Samsung 9 Series (65)" in its evening scans may make certain conclusions about the size of its users living room and income.

C. *Mitigations of BLE Privacy Risks*

We suggest some modifications that would make tracking of the kind we describe more difficult while preserving the promise of innovative devices enabled by the BLE protocol, based on our experience of trying to demonstrate the feasibility of cross-app tracking using BLE devices.

With respect to peripheral device address, we suggest that both Android and Apple should reveal a peripheral uuid rather than the device address to the applications using their Bluetooth APIs, and that these peripheral uuids are randomized at the application and device levels. In other words, different apps should see different peripheral uuids for the peripheral with address P even if the apps are installed on the same phone, and the same app should see a different peripheral uuid for the peripheral P on each phone that it is installed on. Such a change would not significantly impact functionality of BLE devices, but would make peripheral uuids essentially useless for tracking.

Furthermore, we suggest that both Apple and Android increase the barrier for applications to perform scans. On Android, that would mean classifying Bluetooth-related systems permissions as dangerous and requiring per-app user consent for the scans[24]. For Apple, that would mean introducing a user-controlled Bluetooth permission, analogous to the one iOS has for location. Both operating systems could also limit or de-prioritize background scanning, making sure that most of the Bluetooth-related actions happen when the user is using the app. Finally, both could consider introducing a user-visible indicator for when Bluetooth is being used and providing a list of apps that recently used it, similar to Apple's information on location usage by apps.

The Bluetooth Special Interest Group should require and enforce the manufacturers' effective utilization of the "LE privacy" feature.

The proposed modifications, with the exception of background scanning, would not significantly affect true innovations that BLE protocol aims to empower. Thus, their implementation would be a clear win for privacy. They would not fully prevent cross-app tracking, both because once the individual grants an app permission to use Bluetooth for one purpose, there is nothing to stop the app from covertly using it also for tracking, and because of the nature of BLE advertisements that are prone to contain unique data (such as peripheral name) or structured data that is persistent for each peripheral (such as service UUID or manufacturer ID). Thus, to succeed in the cat-and-mouse game of tracking and preventing tracking, additional research is needed to be able to distinguish

[23] Although omitted for brevity, we have experimentally established that *pairing*, a core primitive in LE communication, between a central and a peripheral device, although initiated at the application-level, is done at the device-level in iOS. This fact was already known for Android [11]. Thus, if one application on a phone has paired with a peripheral, then all other applications running on the same phone will see the same peripheral hardware address, effectively making cross-app tracking for users with at least one paired peripheral that can frequently be found near the user, trivial.

[24] This hurdle could be bypassed using forward compatibility, which is an argument for gradually stopping support of older API levels.

Bluetooth-related activity crucial for an app's functionality from activity serving other purposes, such as tracking.

VIII. Related Work

The works most closely related to ours are those of [9], [12], [13], as they suggest that it may be possible to use fixed Bluetooth MAC addresses for tracking peripheral owners. Specifically, [13] presents a detailed analysis of MAC address persistence in advertising packets for a variety of fitness trackers. [12] also focuses on fitness trackers, shows that most do not change their MAC addresses, and further, demonstrates that the BLE traffic between the fitness tracker and the paired central device can serve as a fingerprint. Finally, [9] hypothesizes that other fields besides MAC addresses present in BLE advertisements can also be useful for tracking. We differ from these works in two ways: first, we collect real-world experimental data and perform analyses to support the hypothesis that a particular kind of tracking, cross-app tracking, is indeed feasible in practice. Second, we do so without reliance on an individual owning a particular kind of BLE device such as a fitness tracker.

[14] and [15] aim to mitigate various privacy threats emanating from the BLE protocol but do not address the threat of cross-app tracking. In particular, both focus on preventing unauthorized scanner devices from accessing advertisement information transmitted by devices owned by the user (such as the phone itself), which is a different type of tracking risk than the one we consider.

[16], [17], [18] are works similar to ours but with the tracking risk emanating from the Wi-Fi domain, rather than the BLE domain. [16] presents a tracking attack using information contained in Wi-Fi probes, and is similar to our usage of BLE advertisements. [18] explains Android's failure to assign the appropriate level of protection level to the ACCESS_WIFI_STATE permission and that decision's implications for user privacy, paralleling our observations regarding the BLUETOOTH and BLUETOOTH_ADMIN permissions on Android and no such permission on iOS. [17] finds that a large number of devices from popular manufacturers continue to send Wi-Fi probes with enough information to allow tracking, despite numerous proposed defenses.

[1], [2], [3], [4], [5], [6] present smartphone device fingerprinting techniques for user tracking which exploit hardware imperfections in the sensors introduced during manufacturing. [1] use hardware imperfections in the microphone and accelerometer to fingerprint a device, [3] and [5] rely on microphone and speakers, [2] use diagnostic features such as hardware statistics and system settings extracted using the smartphone's operating system's API, [6] use the personalized device configuration created using the list of applications installed, songs frequently played, language settings, etc. The major difference between our works is in the choice of feature source used for fingerprinting. We rely on BLE sensor data, not previously discussed in this context.

IX. Conclusions and Future Work

We presented a data-driven study demonstrating the feasibility of surreptitious cross-app user tracking using current BLE protocols and Android and iOS APIs. Through our analysis, we motivated the need for modest changes by Bluetooth Special Interest Group, BLE device manufacturers, Apple, and Google that could make such tracking more difficult and give users more control over it, while preserving BLE functionality.

Our future plans are two-fold: first, to disclose the findings to the relevant parties, including the app developers whose apps may be already pursuing such tracking, and work with them to facilitate change. Second, to investigate the feasibility of cross-app tracking without using peripheral uuids, but using other data, optionally contained in the advertising packets (Section III-A) in order to measure the feasibility of tracking and provide additional recommendations for limiting it.

References

[1] H. Bojinov, Y. Michalevsky, G. Nakibly, and D. Boneh, "Mobile device identification via sensor fingerprinting," *arXiv preprint arXiv:1408.1416*, 2014.

[2] A. Quattrone, T. Bhattacharya, L. Kulik, E. Tanin, and J. Bailey, "Is this you?: identifying a mobile user using only diagnostic features," in *Proceedings of the 13th International Conference on Mobile and Ubiquitous Multimedia*. ACM, 2014, pp. 240–243.

[3] A. Das, N. Borisov, and M. Caesar, "Do you hear what I hear?: fingerprinting smart devices through embedded acoustic components," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014, pp. 441–452.

[4] S. Dey, N. Roy, W. Xu, R. R. Choudhury, and S. Nelakuditi, "Accelprint: Imperfections of accelerometers make smartphones trackable," in *NDSS*, 2014.

[5] Z. Zhou, W. Diao, X. Liu, and K. Zhang, "Acoustic fingerprinting revisited: Generate stable device id stealthily with inaudible sound," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014, pp. 429–440.

[6] A. Kurtz, H. Gascon, T. Becker, K. Rieck, and F. Freiling, "Fingerprinting mobile devices using personalized configurations," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 1, pp. 4–19, 2016.

[7] Bluetooth Special Interest Group, *Specification of the Bluetooth System Covered Core Package Version 4.2*, 2014.

[8] J. Leonard, "MAC addresses: the privacy achilles' heel of the internet of things," *Computing*, Nov 9, 2015. [Online]. Available: http://www.computing.co.uk/ctg/news/2433827/mac-addresses-the-privacy-achilles-heel-of-the-internet-of-things

[9] S. Lester and P. Stone, "Bluetooth LE - increasingly popular, but still not very private," 2016. [Online]. Available: http://www.contextis.com/resources/blog/bluetooth-le-increasingly-popular-still-not-very-private/

[10] F. Jacobs, "Apple iOS 9: Security and privacy features," *Medium*, Jun 8, 2015. [Online]. Available: https://medium.com/@FredericJacobs/apple-ios-9-security-privacy-features-8d82d9da10eb#.5zspfi63t

[11] M. Naveed, X.-y. Zhou, S. Demetriou, X. Wang, and C. A. Gunter, "Inside job: Understanding and mitigating the threat of external device mis-binding on android." in *NDSS*, 2014.

[12] A. K. Das, P. H. Pathak, C.-N. Chuah, and P. Mohapatra, "Uncovering privacy leakage in BLE network traffic of wearable fitness trackers," in *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*. ACM, 2016, pp. 99–104.

[13] H. Andrew, P. Christopher, and K. Jeffrey, "Every step you fake: A comparative analysis of fitness tracker privacy and security," Open Effect Report, Tech. Rep., 2016. [Online]. Available: https://openeffect.ca/reports/Every_Step_You_Fake.pdf

[14] K. Fawaz, K.-H. Kim, and K. G. Shin, "Protecting privacy of BLE device users," in *25th USENIX Security Symposium*, 2016.

[15] P. Wang, "Bluetooth low energy-privacy enhancement for advertisement," Ph.D. dissertation, Norwegian University of Science and Technology, Department of Telematics, 2014. [Online]. Available: http://www.diva-portal.org/smash/get/diva2:750267/FULLTEXT01.pdf

[16] M. Vanhoef, C. Matte, M. Cunche, L. S. Cardoso, and F. Piessens, "Why MAC address randomization is not enough: An analysis of Wi-Fi network discovery mechanisms," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security (ASIA CCS)*, 2016, pp. 413–424.

[17] J. Freudiger, "How talkative is your mobile device?: an experimental study of wi-fi probe requests," in *Proceedings of the ACM Conference on Security & Privacy in Wireless & Mobile Networks (WiSec)*, 2015.

[18] J. P. Achara, M. Cunche, V. Roca, and A. Francillon, "Wifileaks: underestimated privacy implications of the access_wifi_state android permission," in *Proceedings of the ACM conference on Security & Privacy in Wireless & Mobile Networks*, 2014, pp. 231–236.