A-SIT Technology
E-Government Innovation Center

28 April 2023 Analysis · Mobile Security · Privacy & Crypto · 16 min read

Comparison of the Fingerprintability of Android and iOS

Gerald Palfinger



Authors	Gerald Palfinger
Version	2023-04-27
Source	https://technology2.a-sit.at/en/publications/comparison-of-the-fingerprintability-of-android-and-ios/
Note	Machine translation.

Key findings

- Both operating systems offer many fingerprintable information sources that allow a device to be recognised across different apps.
- Many categories are available on Android and iOS, such as screen resolution, language and locale, accessibility settings and device name.
- Only iOS offered, among others, keyboard settings and device model number and colour; only Android offered, among others, Bluetooth settings and device lock.
- Through content providers, Android offers an additional source of fingerprintable data that iOS does not have in this form.

- It cannot be said conclusively which system has more fingerprintable sources, as not every method call succeeded.

Tracking users across different applications has enabled application providers to offer the vast majority of their applications for free, as it allows them to target advertising. However, tracking has also proven to be an invasion of user privacy. To counter this, operating system developers have restricted access to unique identifiers in their programming interfaces. However, applications can still combine other non-unique device information to create a unique fingerprint. This report examines what sources of information are available for this purpose on Android and iOS, what types of information are available on both operating systems, and which are different.

1. Introduction

Both on Android and on iOS, extensive tracking of users across different mobile applications is common [1]. Applications and their advertising providers use this tracking to learn more about their users and to serve more targeted advertising. Since over 90% of applications are offered for free, the leading way to monetise applications worldwide is to display in-app advertising [2]. This allows application providers to offer their applications to users free of charge, but the tracking used to deliver relevant advertising severely impairs user privacy [3]. To counter this, Apple has taken measures in recent versions of iOS to restrict tracking by third-party applications and their tracking libraries. Developers must now obtain the user's consent using the App Tracking Transparency framework [4]. They must also state which types of information they use to track users. Technically, these changes prevent the collection of the unique advertising identifier when no consent has been given. However, a recent study [5] has shown that a large number of applications still collect, or even begin to collect, other data from the operating system that can be used to create a fingerprint of the device. A similar restriction of access to unique identifiers was also introduced on Android with version 10 [6]. As access to unique identifiers is restricted, the technique of device fingerprinting is therefore becoming increasingly relevant. In browsers, for example, where there are no unique identifiers that can be used to identify users, fingerprinting is already widespread [7]. In essence, browser fingerprinting aims to collect as much unique data about the environment as possible in order to distinguish users. A similar methodology can be applied on iOS [8] as well as on Android [9] to recognise devices. However, which types of information are available for this purpose, and whether and how they differ between the two operating systems, has not yet been systematically examined. This report addresses that question.

2. Related Work

In [8] it was shown that it is possible to create fingerprints of iOS devices from values obtained from the programming interface. The authors manually selected 29 information sources and built an application that collected data from these sources on the devices of different users. Although the values obtained from these information sources were not unique on their own, their experiments showed that combining the values made it possible to re-identify the devices in their dataset with high accuracy. Similar to that study, [9] also showed that a device can be uniquely identified on Android through a combination of non-unique

information. The authors selected 38 different information sources, which they again collected from different devices. To re-identify the devices from the collected datasets, they used various algorithms and then compared their accuracy. They showed that re-identification is possible with high probability.

In contrast to the two previous studies, in which the authors selected a set of information sources, [10] examined the types of information used by real tracking libraries on Android. To find these libraries, they relied on the list of information sources provided in [9]. In addition, they extended it with the information sources used by two already known fingerprinting libraries. Using these information sources, they found six similar libraries that create device fingerprints by means of static analysis. They also concluded that different techniques are used for fingerprinting smartphones than for fingerprinting browsers. A more systematic approach was shown in [11]. For this, a framework called AndroPRINT was developed to collect data from the Android API. The framework obtains data from fields, methods and the Android-specific content providers. Using AndroPRINT, it was shown that a large number of these sources provide fingerprintable information. The analysis also showed that vendor customisations can reveal personal data and provide additional information sources for fingerprinting, including some unique and user-specific identifiers. In [12], this principle was transferred to the iOS operating system. An automated framework was used to search the programming interface of the operating system for fingerprintable information sources. It was likewise shown that there is a large number of sources that may be suitable for fingerprinting. Based on these two frameworks, this report compares which information sources exist on both operating systems and which could only be found on one of the two.

3. Methodology

A smartphone application was created for both Android and iOS to systematically collect data that is available to third-party applications via the programming interface. Using reflection, the application instantiates all classes on the device. For each of the created objects, all instance methods of the class are called and the return values are stored. In addition, all fields or properties of the instantiated classes are retrieved. Finally, every class method is called using the class descriptor. The smartphone application is used to collect data from several devices. To simulate two different application developers collecting the data, we run the smartphone application twice, using different developer profiles and bundle identifiers on iOS and different signing keys and package names on Android. The framework then prepares the collected data for cross-device and cross-platform analysis. For this, the data is cleaned and formatted uniformly. Finally, the data is analysed to identify fingerprintable information sources in the programming interfaces. First, data from the same information source that differs between the two runs on the same device is discarded, because it is not robust across different application vendors or application restarts. The remaining data is then compared across devices. Data sources that differ between devices are considered fingerprintable. For the comparison across the two operating systems, they are categorised by the type of information obtained. The following two sections explain how the search for potentially fingerprintable information differs between the two operating systems due to technical circumstances. The analysis of the collected data is covered afterwards.

3.1. Android

Before information can be extracted from the Android programming interface, certain information must be parsed from the documentation, because it cannot be retrieved while the smartphone application is running. This is the list of classes of the programming interface. These are needed to create objects of all

classes of the programming interface. The names of method parameters must also be extracted from the documentation, because during execution they are replaced by generic names such as arg_0, arg_1,....

Once this information has been obtained, the smartphone application can carry out the actual data collection. Using the list of parsed classes of the programming interface, objects of these classes are created by means of reflection. Methods are then called on these objects. If a method requires parameters, the application creates them beforehand. After the method call, the return value is stored for later analysis. If a non-primitive object is returned, it is additionally used to call further methods.

Besides calling methods, all accessible fields of the programming interface are also queried. The objects created for the method calls and the non-primitive return values of the methods are likewise used for this query. The values of these fields are also stored for later analysis.

Android has a concept that is specific to it: content providers. A content provider is an abstract interface for accessing data. Similar to a database, the contents can be queried in the form of rows and columns using a cursor. A content URI is required for this. To obtain the content URIs of the content providers supplied by the system, we analyse every queried field and every returned string value during the method calls. If a string begins with the prefix content://, the URI is stored for later retrieval. Once the method calls and field evaluations are complete, the content of all detected and accessible content providers is read out by iterating over all rows and columns. Like the return values and fields, the data read out is stored for later analysis.

3.2. iOS

The framework for collecting the raw data on iOS is divided into two main components: the backend and the smartphone application. In essence, the smartphone application retrieves data from the programming interface via reflection, similar to Android. As on Android, methods are called automatically and properties (similar to fields on Android) are read. However, due to the low-level nature of Objective-C, failed method calls or object instantiations can lead to memory problems that cause the smartphone application to terminate. To solve this problem, the backend monitors the execution of the smartphone application and restarts it when necessary. Before the restart, methods, classes and properties that caused the problem are excluded from being called, instantiated or read again. Similar to Android, the backend retrieves in advance some specific information that is not available at runtime. In the case of iOS, this is the concrete type of method parameters, which cannot be determined at runtime. Finally, the backend transfers the collected data from the smartphone application for analysis. On iOS, this data is read from the system log, which makes further structuring of the collected data necessary.

The backend initially stores the output of the smartphone application in the system log unchanged. All outputs of the smartphone application are retained in the stored log, for example also the outputs of called methods. The collected data must therefore be cleaned before analysis. This is done by our analysis application. It collects the return values from the system log and discards superfluous outputs. It also converts the results into a predefined format. This was defined as CLASS.METHOD:::RESULT or CLASS.PROPERTY:::RESULT and thus matches the output of the Android application. Since the result can contain line breaks, these are replaced by the string -NEWLINE-, as on Android. With these steps, the results can be compared across different devices.

3.3. Analysis of the Raw Data

Data collection is carried out twice on each smartphone. In between, the smartphone application is removed and the device is restarted. After the application has been removed, it is reinstalled with a different signing certificate and a different bundle identifier. The second collection phase is then started. This elaborate process is carried out to simulate data collection by applications from different developers. In this way, the information sources that differ between two different applications, and are therefore not suitable for creating a cross-application fingerprint, are eliminated. For example, false-positive information such as the installation directory of the application is eliminated, as it contains a random value that differs for each application. Developer-specific identifiers such as the UIDevice.identifierForVendor property on iOS or the Settings.Secure.ANDROID_ID value on Android are likewise excluded by this approach. Finally, this measure also eliminates all temporary values that could change due to an application restart or a device restart. All remaining values, that is, those that do not differ between the two runs on the same device, are then stored. These are collected for all examined devices and compared in the next step, the analysis.

During the analysis, the collected, cleaned and ordered values of one device are compared with all other examined devices. Methods that could only be called on one device, or properties or fields that could only be retrieved on one device, are removed because there is no value to compare them with. All remaining values, that is, those that could be retrieved on at least two devices, are then compared automatically. All values that are identical on all analysed devices are removed. All values that differ on at least one device are retained and their source is marked as potentially fingerprintable. These values are then reviewed and categorised manually to make them comparable between iOS and Android. The results of this analysis can be found in the next chapter.

4. Results

To collect the fingerprintable information sources, the created smartphone applications were run on Android and iOS devices. Pixel 4 devices were used on Android, while iPhone 11 devices were used on iOS. Table 1 shows the results of the evaluation. The sources found were divided into stable and unstable sources. Unstable sources can change, for example, when the device is restarted. Since smartphones can be in operation for days or even weeks, such information could still be used to create fingerprints of a user.

Many of the categories of information sources found are available on both Android and iOS. The unstable sources available on both operating systems include network information and information about the boot process. However, considerably more information about connected networks is available on Android. On iOS, the boot time can be retrieved, while on Android the number of boot processes can be determined.

In addition, a considerably larger number of more stable information sources was found on both operating systems. The sources present on both operating systems include information about the size of various user interface elements as well as information about the screen resolution. The language and various locale data can also be determined through a large number of sources. The date and time format can also be retrieved on both systems. Information about the configured currency is likewise available. Various accessibility settings are present on both systems, which either affect the user interface or can be retrieved directly. Settings that affect the colours or the appearance of the user interface can likewise be detected everywhere. The network operator in use can also be retrieved on both operating systems. Information

about the input method can be retrieved on both Android and iOS. Certain details about system applications can also be determined. Some settings for system sound and Do Not Disturb can likewise be retrieved. The device name defined by the user could be detected on both systems.

Details about the keyboard in use and keyboard settings could only be retrieved on iOS. The model number and the colour of the device could likewise only be detected on iOS. The user persona and further identifiers existed only on iOS. No direct identifiers were detected on Android. The activation status of personalised advertising and UI effects could also only be detected on iOS.

There were also information sources on Android that could not be detected in the same way on iOS. These include various Bluetooth settings, activated networks, ringtones and notification settings. Various display settings, sound effects and information about the device lock in use and the password complexity could only be found on Android. In addition, Android-specific features could be detected, such as widgets, screen savers or the activated personal assistant. Furthermore, concrete timestamps and the customisation status could be detected.

Table 1 Information sources found on Android and iOS when comparing devices of the same model (CP = Content Provider).

Category	Android			iOS	
	# Methods	# Fields	# CP	# Methods	# Properties
Unstable					
Network information	44	10	4	4	-
Boot time	-	-	-	2	-
Boot count	-	-	2	-	-
Other	-	1	-	-	-
TOTAL	44	11	6	6	0
Stable					
UI size & screen resolution	90	40	2	114	139
Language and locale	180	8	3	89	31
Keyboard settings	-	-	-	54	33
Accessibility settings	9	3	21	39	22
Date & time format	2	1	-	16	21
UI colours & style	21	3	-	7	4
Model number	-	-	-	5	5
User persona and other identifiers	-	-	-	6	4
Currency information	22	-	-	5	5
Network operator and MMS setting	29	2	6	5	2
Input method	16	1	3	6	-
(Deleted) system applications	6	-	1	6	-
Miscellaneous settings	-	-	109	4	-
Device colour	-	-	-	1	3
Application information	-	-	-	-	3
Other	11	3	1	2	1
Personalised advertising	-	-	-	1	-
UI effects	-	-	-	1	-
System sound	5	-	-	1	-

Category	Android		iOS		
User-defined device name	2		2	-	1
Bluetooth settings	17	-	1	-	-
Activated networks	23	-	2	-	-
Notification settings	4	-	6	-	-
Display settings	5	1	8	-	-
Ringtones	5	1	-	-	-
Sound effects	-	-	6	-	-
Device lock	3	-	-	-	-
Widgets	2	-	-	-	-
Timestamps	-	-	2	-	-
Password complexity	1	-	-	-	-
Customisation status	-	-	1	-	-
Screen saver	-	-	1	-	-
Activated assistant	-	-	1	-	-
TOTAL	453	63	176	362	274

5. Conclusion

For this report, potentially fingerprintable information sources were collected from Android and iOS devices. The collected sources were then categorised by the type of information obtained. It turned out that many categories of information sources are available on both Android and iOS. Many of the information sources that could only be found on one of the two operating systems come from features that do not exist in the same form on the other operating system. However, through content providers, Android offers an additional source of fingerprintable material that does not exist in this form on iOS. Since each method can only be called with a limited number of different parameters, and since not every method call or object instantiation could be completed successfully, it is not possible to conclusively say which of the two operating systems has more fingerprintable information sources. However, on both operating systems there is a large number of fingerprintable information sources that make it possible to recognise a user across different applications.

References

[1] K. Kollnig, A. Shuba, R. Binns, M. V. Kleek und N. Shadbolt, „Are iPhones Really Better for Privacy? A Comparative Study of iOS and Android Apps,“ Proc. Priv. Enhancing Technol., pp. 6–24, 2022.

[2] Statista, „Mobile app monetization - Statistics & Facts,“ Statista, 4 8 2022. [Online]. Available: https://www.statista.com/topics/983/mobile-app-monetization/#topicOverview. [Zugriff am 24 04 2023].

[3] I. Shklovski, S. D. Mainwaring, H. H. Skúladóttir und H. Borgthorsson, „Leakiness and creepiness in app space: perceptions of privacy and mobile app use,“ Conference on Human Factors in Computing Systems, pp. 2347-2356, 2014.

[4] „App Tracking Transparency,“ Apple Inc., [Online]. Available: https://developer.apple.com/documentation/apptrackingtransparency/. [Zugriff am 27 04 2023].

[5] K. Kollnig, A. Shuba, M. V. Kleek, R. Binns und N. Shadbolt, „Goodbye Tracking? Impact of iOS App Tracking Transparency and Privacy Labels,“ Conference on Fairness, Accountability, and Transparency, pp. 508-520, 2022.

[6] Google, „ Privacy in Android 10,“ [Online]. Available: https://developer.android.com/about/versions/10/privacy/. [Zugriff am 25 04 2023].

[7] U. Iqbal, S. Englehardt und Z. Shafiq, „Fingerprinting the Fingerprinters: Learning to Detect Browser Fingerprinting Behaviors,“ Symposium on Security and Privacy, pp. 1143-1161, 2021.

[8] A. Kurtz, H. Gascon, T. Becker, K. Rieck und F. C. Freiling, „Fingerprinting Mobile Devices Using Personalized Configurations,“ PoPETs, pp. 4–19, 2016.

[9] W. Wu, J. Wu, Y. Wang, Z. Ling und M. Yang, „Efficient Fingerprinting-Based Android Device Identification With Zero-Permission Identifiers,“ IEEE Access, pp. 8073–8083, 2016.

[10] C. F. Torres und H. Jonker, „Investigating Fingerprinters and Fingerprinting-Alike Behaviour of Android Applications,“ European Symposium on Research in Computer Security - ESORICS, pp. 60-80, 2018.

[11] G. Palfinger und B. Prünster, „AndroPRINT: Analysing the Fingerprintability of the Android API,“ Availability, Reliability and Security - ARES, pp. 94:1 - 94:10, 2020.

[12] G. Palfinger, „OCScraper: Automated Analysis of the Fingerprintability of the iOS API,“ SECRYPT, 2023.