

The All Seeing Eye: Web to App Intercommunication for Session Fingerprinting in Android

Efthimios Alepis and Constantinos Patsakis✉

Department of Informatics, University of Piraeus
80, Karaoli & Dimitriou, 18534, Piraeus, Greece.
talepis@unipi.gr, kpatsak@unipi.gr

Abstract. The vast adoption of mobile devices in our everyday lives, apart from facilitating us through their various enhanced capabilities, has also raised serious privacy concerns. While mobile devices are equipped with numerous sensors which offer context-awareness to their installed apps, they can be also exploited to reveal sensitive information when correlated with other data or sources. Companies have introduced a plethora of privacy invasive methods to harvest user’s personal data for profiling and monetizing purposes. Nonetheless, up to now, these methods were constrained by the environment they operate, e.g. browser vs mobile app, and since only a handful of businesses could have access to both of these environments, the conceivable risks can be calculated and the involved enterprises can be somehow monitored and regulated. This work introduces some novel user deanonymisation approaches for device fingerprinting in Android. Having Android AOSP as our baseline, we prove that web pages, by using several inherent mechanisms, can cooperate with installed mobile apps to identify which sessions operate in specific devices and consequently to further expose users’ privacy.

Keywords: Android; Privacy; Deanonymization; App collusion

1 Introduction

The unprecedented growth of mobile usage has radically transformed our daily lives. Besides the great advances in our communications, mobile devices have changed the way we create, process and consume information, as they realise pervasive and ubiquitous computing. Among others, one of the most significant emerged changes is how we value information. The fact that people are constantly and effortlessly connected to the Internet via devices which empower people’s unobstructed communication, information flow and entertainment, in many occasions results in disregarding or underestimating the value of the information they consume and offer to third-parties.

As far as information offering is concerned, the value of the provided information to third-parties is most of the cases considerably high, something that is not

always understood by the users. For instance, one might share his location with an app or a web page neglecting the fact that this single piece of information also consists of a very sensitive piece of data which can be exploited for various purposes. Indicative uses for such location sharing could be the recommendation of other users in proximity for communication purposes, or even for sharing a ride. Aggregating location data from numerous users can provide real-time traffic analytics or insight about resource requirements in a smart city. Apparently, this information can stimulate businesses' prosperity by enabling the implementation of further customer-centered services. Therefore most companies are striving to extract from users as much information as possible.

While service personalisation can be considered as a noble cause, companies tend to exploit data further for profiling and targeted advertising, tactics that can expose users to many privacy hazards. This trend is highlighted by the fact that many companies are providing APIs which harvest user data to create fine grained user profiles, containing a lot of sensitive user information. Such practices have also led to the introduction of methods such as browser and device fingerprinting. Nonetheless, thus far mobile apps and web pages are considered as two diverse ecosystems as they refer to two discrete software environments with radical differences in their information flow and data usage. This distinction works for the benefit of users' privacy, since it allows some parts of their activities to remain isolated and hence private. For instance, it prevents an app from knowing which web pages a user visits, or a web page from knowing which apps a user is using and when. On the contrary, enabling access between these two environments could allow for a web page to communicate with an installed app to recover further personal and sensitive information from local files or sensor measurements, and hence to further reveal one's interests.

The goal of this work is to illustrate that there are currently several means to realise user identification in Android, regardless of the environment a software module is operating on. Despite the privacy hesitations of people towards the well-known tech giants or independent browsers, we provide some concrete examples proving that an "All Seeing Eye", a software entity able to monitor users' actions across both the web and the application environments, can be easily created. Such an entity, in the form of an online database equipped with some additional services can correlate information from web pages and mobile apps in order to identify individuals. After a thorough investigation in the related scientific literature and to the best of our knowledge, the authors of this paper have concluded that this problem has been so far partially studied, as current literature is focused on methods which examine each software ecosystem independently and not both of them as a whole. In fact, the proposed methods in this work can be considered as an extension of device fingerprinting as they do not solely depend upon unique characteristics of device components or hardware identifiers. We name these methods as "session fingerprinting" since their goal is to reveal whether web-browsing and software sessions operate simultaneously in a device and identify the user.

The generic concept of this work, in a simplified form, is illustrated in Figure 1. Each side of this figure is dedicated to the two software “ecosystems”, namely web pages and mobile apps. Obviously, there is a crosscut from the OS, namely Android, since it manages calls from both ecosystems in a mobile device, as well as from the browsers which by their definition as applications belong in both ecosystems. The “All Seeing Eye” acts as a Command and Control, C&C, server which collects information from web pages and apps, correlates it and transmits “commands” and the corresponding information to both sides. The commands may range from “retrieve a list of installed apps” and “scan local storage for files containing X”, to “display ad Y” or “application Z send webpage data W”. Therefore, the “All Seeing Eye”, as the orchestrator of all performed actions by apps and web pages, can ultimately reveal user identities.

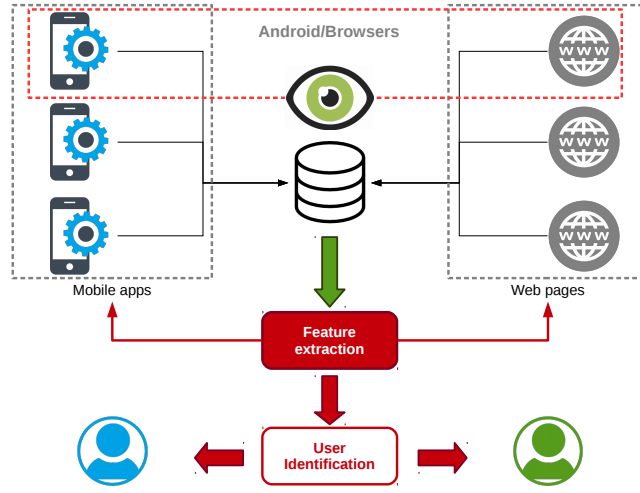


Fig. 1: Basic concept

While similar attempts have been made in the past, it is rather important to note that methods trying to escape the browser’s environment without user’s consent are considered to be malware and usually exploit browser’s vulnerability. Especially in the case of Android, passing a single bit of information from a benign browser to an app is rather difficult, given that it has not only to bypass the browser sandbox but additional obstacles due to Android’s security model which will be discussed later on.

2 Session Fingerprinting

When someone browses the Internet, his session is considered anonymous unless he has logged in a web page. While this anonymity is very convenient for ensuring

user's privacy, companies strive to find ways to bypass it and to profile users. Frequently, as the benign goal behind these actions is regarded the adaptation and/or personalisation of a web page according to the corresponding user profile, which translates to better usability and increased content quality, which in turn may increase both views and viewers. In most of the cases, this personalisation targets the advertising industry, since by deanonymising an individual a business is able to display ads tailored to users' preferences and therefore to increase both business' and service providers' profits.

One of the most widely used methods in achieving this is browser fingerprinting, which tries to deanonymise users by exploiting noticeable differences in the usage of different browsers such as the underlying OS, user agent, browser version, monitor size, or even installed fonts and plugins [10, 15, 16]. More advanced methods go a step further by exploiting device specific variations to identify individual devices. For smartphones, it has been shown that sensors, such as accelerometers, or speakers and microphones may have unique characteristics which differ, not only across models, but also across the devices within they operate due to calibration errors and frequency distortions [9, 20, 6].

While these methods have been proven efficient in many cases, they are usually subject to errors and software updates which could render a previous fingerprint useless. For instance, a browser update may change the user agent or the fonts, making impossible its linking to the previous fingerprint. However, what a company actually needs is to be able to correlate information with other affiliated parties in order to determine whether the user has been simultaneously operating another *session*. As a typical example can be regarded the parallel usage of a web page and a mobile app. Note that while the latter means that the app is running in the background, it is a typical situation in almost all mobile OSes. In this regard, both parties should try to create a unique ID for each session and also communicate it with each other in order to deanonymise the user. We name the methods for extracting these IDs as *session fingerprinting*.

Apparently, these methods depend on the existence of cooperating apps in the mobile device. We argue that this is a weak assumption as the considered adversary in this work is mainly an ad network. Due to the prevalence of the freemium model in Android, most applications are free and, most of the times, they come with at least one preinstalled ad component. However, our requirements do not imply escalated privileges, hence the resulting applications are easier to be accepted by the users.

Although both browsers and the Android OS have such privileges, namely are able to deanonymise the users and have data about them coming from multiple sources, the level of trust a user has to both of them is the ultimate one. User choose them because they trusts that they will act honestly and they will protect them from threats and, above all, they will not stalk them. Moreover, it is important to highlight that despite the OSes restrictions, the different existing ad frameworks may indeed perform user profiling, and in many cases, a quite intense one, but still they cannot escape the browser or the app ecosystem within they operate. Nonetheless, they are continuously acting more rogue,

despite their sandbox environment [12]. Stevens et al. [19] found that some ads would use undocumented permissions, such as read/write to calendar or access location and camera. Grace et al. [13] found that around half of them would probe the corresponding apps to determine whether they could abuse them for harvesting sensitive user information. More recently, it has been documented that ad networks are using commercial ultrasonic tracking technologies like SilverPush, Lisnr and Shopkick to determine either users' location, or the commercials they are watching [2]. Notwithstanding their invasiveness, to the best of our knowledge, none of them has been able to pass information from a browser to an app within the Android system. Instead, this kind of communication, whenever reported, was strictly among apps that used the same ad network.

3 Problem setting

In principle, the information that can be collected from a web page is derived solely via the launched browser and is strictly limited to the browser's environment. Any attempt to access resources beyond the browser sandbox is considered as a security violation and therefore is characterized as malicious. To this end, browsers allow very limited exposure of user data to a web page. For instance, a web page cannot read from or write to the storage of a mobile device unless this action is user initiated. To overcome these restrictions, adversaries may resort to browser extensions [14] which provide even more capabilities. On top of that, nowadays, due to the growth of mobile devices, several standards, like HTML5, have been introduced to allow browsers to access additional resources, such as location, camera, or microphone, upon direct and explicit user consent. As a result, web pages have a growing set of capabilities, yet quite limited in comparison to apps.

On the other hand, Android apps reside on a different environment. Contrary to web pages, mobile apps "live" on the operating system and thus have more direct access to the hardware. Again, their access is limited according to the granted privileges from the users plus their scope is more fixed as they fulfill specific user needs. Due to this restriction, apps cannot determine which web pages a user visits. A critical distinction between Android apps and web pages is that apps always pass through an "installation" process. This step represents a user acknowledgment regarding the specific resources an app is allowed to use inside the environment it is executed. Notably, since Android Marshmallow users may grant and revoke permissions to specific resources, like camera, microphone, location etc, which are called "dangerous" and may hinder security and privacy issues. On the contrary, this is not the case for web pages where users do not have preconditions for visiting them. In many instances, users would like to be able to use "one-time apps" to accomplish specific tasks like using a retailer's app when browsing his web page. To address this need, Google recently introduced *instant apps*, which do not require installation, and have more permissions and/or capabilities than common web pages. However, instant apps, like web pages, are also restricted from accessing hardware identifiers to prevent user profiling.

Key differences in the number of capabilities of Android apps, both native and instant, and web pages are illustrated in Table 1. As expected from the earlier discussion, it can be easily noticed that installed applications have far more access to device resources than web pages, since users install apps granting themselves the corresponding permissions. On the contrary, due to the nature of the Web, users may visit a large number of different web pages on a daily basis, without knowing their quality, intentions, source or content. Hence, both Android and browsers make significant efforts, e.g. running in a sandbox environment, towards protecting users from malicious web page behaviour. Unquestionably, if an app had been able to communicate with a web page without restrictions, the entire underlying security infrastructure would have been rendered useless. In fact, even the most widely used apps in Android are not able to communicate directly with their web page. For instance, in the case of three well-known and widely used apps, Facebook, Instagram and Twitter, they do not transfer information to their corresponding web page or any other cooperating web page when a user has logged in the app. Instead, a “Connect with Facebook/Twitter” button usually appears, requiring further user interaction and most importantly being realized by users. Evidently, had these apps been able to transfer this kind of information to the browser, they would have done it already long time ago, not only for facilitating users, but for increasing further the amount of collected user data and the quality of provided services.

Creating a mechanism being able to transmit an identifier from the browser to a cooperating installed app in a user’s device, or vice versa, would allowed for the installed app to identify the individual who visited the cooperating web page and subsequently, that web page would be instantly granted access to the same resources as those of the installed app. Further analyzing this, after both parties, namely web pages and apps have identified themselves lying in the same user’s device, they are able to create a covert channel. In the least sinister scenario, a web page cooperating with an app would be able to access a user’s contacts, SMS messages or even storage and microphone, without obtaining user’s consent, and would displayed ads perfectly tailored to the user’s profile, albeit violating his privacy. However, in a true malicious scenario, user data would be harvested by web pages and personalized exploits would be pushed to users’ devices to further exploit their personal data while they surf in the WWW.

In most of the Android cases documented by researchers, information is leaked from one app to another through a covert channel [11, 8]. Although Rushanan et al. in [18] achieve a goal similar to ours, their study concerns only the desktop environment. Their approach consists in exploiting the Web Workers API in order to increase the CPU and memory utilization. By monitoring both CPU and memory usage, they manage to pass messages from a web page to an app in a desktop computer. However, this attack scenario is not possible in an Android device. For devices up to Marshmallow, while apps could monitor the `/proc/` folder and extract some information about memory usage, the recovered information is far from being considered fine-grained and does not include CPU usage. With the introduction of Nougat, apps are allowed to only access the con-

	Native Apps	Instant Apps	Web Pages
Access Device Identifiers	✓		
Device External Storage	✓		
Push Notifications	✓		
List of Installed Apps	✓		
Access Body Sensors	✓		
Direct Communication with Installed Apps	✓		
Receive Broadcasts from OS or 3rd party Apps	✓		
Run on the Background	✓		
Change Device Settings	✓		
Access User Calendar	✓	✓	
Access motion sensors	✓	✓	
Access Contacts	✓	✓	
Access Phone Calls	✓	✓	
Access Sensors	✓	✓	
Access environmental sensors	✓	✓	
Access Location	✓	✓	✓
Access Microphone	✓	✓	✓
Access position sensors	✓	✓	✓
Access Camera	✓	✓	✓
Access Internal Storage (own storage)	✓	✓	✓
High precision timestamps	✓	✓	✓

Table 1: Capabilities of apps and web pages.

tents of their own `/proc/PID` folder (<https://developer.android.com/about/versions/nougat/android-7.0-changes.html>), so this method does not work any more for AOSP. The only other alternative for an app to have this kind of access is to request the system-level permission `PACKAGE_USAGE_STATS` (<https://developer.android.com/reference/android/app/usage/UsageStatsManager.html>). The fact that their attack does not apply for passing messages in Android is also proved by the authors' statement that in Android they managed just to launch a resource depletion attack against the browsers. Moreover, the aforementioned restrictions in Nougat prevent apps from accessing `/proc/net` which could otherwise reveal the domain names but not the full URL a user has visited.

Notably, developers in many occasions, despite Google's recommendations (<https://developer.android.com/training/articles/user-data-ids.html>), use `ANDROID_ID` as a unique identifier. To restrict this, Google required for apps to request the dangerous permission `READ_PHONE_STATE` (<https://developer.android.com/reference/android/Manifest.permission.html>). Clearly, since this ID is unique, installed apps may identify instances and correlate users and behaviours. Since such actions violate user privacy, even though they are performed locally only among installed apps, in the latest preview of Android O, Google decided to block this behaviour so that each app receives a different `ANDROID_ID`. More precisely, in Android O for each combination of application package name, signature, user, and device we end up with a different `ANDROID_ID` (<https://developer.android.com/preview/behavior-changes.html>). To further support users controlling their unique identifiers, Google has recently announced the new changes coming in Android O [5], regarding device identifiers. In this regard, Android O is limiting the use of device-scoped identifiers that are not resettable and is also updating the way that applications request account information and providing more user-facing control. The latter signifies that Google is not only aware of such deanonymization issues, but is also constantly working on refining its platform to mitigate these threats and restrict unauthorised and unregulated app to app communication, let alone web to app communication.

4 Intercommunication between apps and web pages

In the following subsections, we provide a set of concrete examples as Proofs of Concept, which showcase how apps and web pages can mutually create and consequently transmit unique IDs that allow them to link their usage and to communicate sensitive attributes to each other, realizing what the authors of this paper have introduced as "session fingerprinting". The authors of this paper have responsibly disclosed the mentioned security issues described in the next subsections, regarding unauthorized communications between apps and web pages.

4.1 Location

Location awareness has undoubtedly increased the potential of many applications since it allows them to adapt accordingly and render their information

based on location specific criteria, drastically improving e.g. user recommendations. Obviously, location is a sensitive piece of information as it can disclose many private attributes, ranging from work and residence location, to entertainment preferences and political/religious beliefs if correlated with other sources of information. Therefore, mobile OSes allow applications to access location data only if the user grants a corresponding permission. In Android this permission is provided either as *Fine* or as *Coarse* location.

Similarly, beyond the support for media in HTML5, the standard enables web pages to access user location. Since this information is sensitive, the browser specifically requests for user permission to be granted, even though this kind of information can be used for other purposes as well. Once the browser gets access to user's location, the response contains apart from the longitude and the latitude, the accuracy and the timestamp [17] as well. Moreover, depending on the implementation, it may also return other values, such as heading and speed. Interestingly, in our research we have come up with proofs that this information can be correlated with location data information from an Android app. More precisely, while an Android app could monitor a user's location and is able to correlate the coordinates with the ones that are received from a web page, one could argue that since these requests to location data are not made simultaneously, from the browser and the app, the actual identity of the user is not disclosed. This argument can be clearly supported either because other nearby users may be also implicated or because the web page gets this information only once. However, practically this is not the case. For reasons such as decreasing battery consumption, since the usage of the GPS is rather greedy, Android app developers may choose to use the "last known location" feature through `getLastLocation` of `LocationServices` which fetches the location from its cache [1]. Then, based on the accompanied timestamp the developer can determine whether he needs to request a new reading or not. Yet, what seems quite interesting in this case is that our findings reveal that by accessing the device's last known coarse location from an app, we may end up having data about the precise last location request made by a web page. It should be emphasized that "coarse" location is the one that should be used here, since "fine" location is accessed exclusively by android apps and not web pages and hence is not suitable for our method.

Apparently, if a mobile app monitors the last known location and its corresponding timestamp determined by the `Network Location Provider` and communicates this information to the All Seeing Eye, the latter is able to determine whether it coincides with user's coordinates and timestamp received from a web page. Beyond a doubt, this combination of data is quite "unique". Once a correlation is found, the All Seeing Eye can create a covert channel that will serve both the app and the web page to exchange data regarding this session.

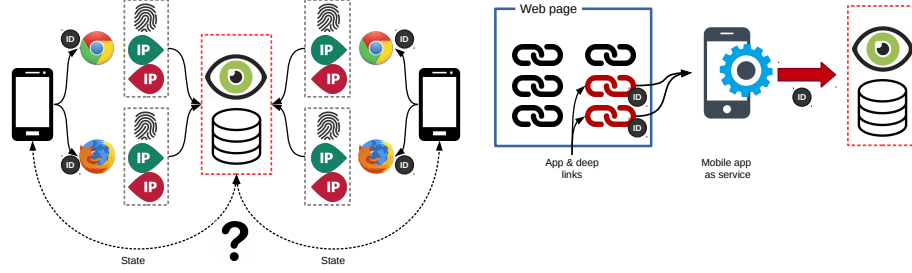
4.2 Browser fingerprinting

In the previous example a dangerous, yet very commonly used permission, namely location, was used to identify a user. Nevertheless, one could achieve the same

result without such permissions. A more stealth method is to utilize browser fingerprinting. To this end, we assume that the victim has installed an application which does not request any dangerous permission. According to the Android permission model, such applications are allowed to list all the installed applications in a device, hence the adversary also has knowledge of all the installed browser applications. This way, the app can subsequently open all the available browsers through intents and point them to a desired URL in order to obtain a fingerprint from them. Note that as of Nougat, an application cannot determine which the foreground application is, a piece of information that would have been very valuable for the adversary, however, the authors have already notified Google of a new method to achieve this in all versions prior to Nougat (Android Issue no 23504, triaged). Each time a browser is fingerprinted by the app, a random nonce is created and is sent in the web page request allowing the adversary to determine to whom its fingerprint belongs. This kind of attack utilizes malicious intents, while for “covering traces” purposes the cooperating malicious web pages could redirect to a commonly used web page (e.g. a search engine), after accomplishing the ID exchanging job. A scenario where no intents are needed also exists, where a malicious app may use its native webview component in order to accomplish the aforementioned task.

Since mobile devices have less “unique” characteristics compared to personal computers, an extension to browser fingerprints is to additionally use even more mobile device characteristics, further conforming to the “session fingerprinting” proposed term. Indeed, both a mobile app and a web page can obtain knowledge about the internal and the external IP of the mobile device. For the former one could potentially use WebRTC [4] which is known to leak several pieces of private information [3]. Therefore, when a user visits a web page, the web page queries the “All Seeing Eye” to determine if someone with the specific browser fingerprint, public and local IPs has a cooperating app running at this timeframe. In general, the chances of this query returning more than one result are slim. Nonetheless, in a corporate environment where many people might have mobile devices of the same model, some instances may exist. In such environments one could have two identical devices with the same internal IP, if e.g. two users with the same smartphone model use the corporate WiFi on different floors or departments. To further reduce the query results, the “All Seeing Eye” could request the state of each device. In this case, additional information that can be cross-checked between apps and web pages include, but are not limited to, the following: battery information (both state and charging level), interval since device’s last noticeable movement (this could be determined e.g. via accelerometers), interval since last proximity (via proximity sensor), light measurements, positioning (e.g. facing up or down), or even some connection statistics such as `downloadMax` (one of the new features of HTML5 through the Network Information API [7]). From this information one can easily determine which user is using a smartphone at a specific timeframe and essentially to eliminate the possibilities of having false positives. The process is illustrated in Figure 2a. In this figure, we may notice that both web pages inside browsers and also web pages

inside applications’ webview components are able to collect unique fingerprints, namely internal and external IPs, accompanied by information regarding the devices’ state and communicate them to the “All Seeing Eye” which will then be responsible for finding exact matches between them.



(a) Identification through browser fingerprinting. (b) Identification through app and deep links.

Fig. 2: User identification methods.

4.3 App and deep links

Most users install plenty of apps on their devices, even though many of them might have some overlapping functionality. To facilitate user interaction between websites and Android applications the Web Intents framework was introduced, allowing a developer to specify how a hyperlink is handled on the user device, e.g. open the phone to dial up an already prepared number, or use Skype for a specific contact. However, Android supports further features through *App* and *Deep links*. The concept behind both of these types of web links is to open specific apps depending on the link. As an example, Facebook and Twitter apps are triggered when the user taps on a link referring to content of the corresponding site.

Interestingly, this kind of functionality is automatically activated when a user installs an application which provides such features, without requesting any user approval. Moreover, Android activities may run on the foreground in a “hidden” mode, either by using transparent themes or by utilizing floating zero-sized activities. Practically, this creates a hidden communication channel between web pages and apps that can be used to identify users as illustrated in Figure 2b. We assume that an app is installed in a user’s device having at least one “browsable” (declared inside Apps Manifest file) activity in order to enable “cooperation” with web pages. On the other side, web pages embed some special “intent” hyperlinks which also have the ability to carry a random ID,

different for every interaction. Once a user taps in one of these links, the ID is bundled and transferred to the app, using the “getExtras” method inside the “Intent” Android class. As a final step, once again the data is communicated to the All Seeing Eye, deanonymising the user. This kind of “interaction” can be seen as a directed web-to-app communication, where an app has the ability to be reached from web pages. At the same time one or more web pages may utilize this “functionality” by transmitting an ID which is essential for the entire described scenario. Next subsection describes how this local channel communication can be achieved through the opposite direction of interaction.

4.4 Direct App to Web communication

Following the same logic as in the previous subsections, an app is also able to directly reach a web page through device’s installed browser. Once again, Android Intents are deployed in order to launch an installed browser and to pass a URL. The cooperating app is able to inject one or more string values inside a URL as parameters, which in our case is a simple, randomly generated ID and correspondingly fire a malicious intent towards a browser. As a next step, the loaded web page extracts the ID from the URL’s “location search” property and thus a local covert channel between the app and the launched web page is created. Of course, the web page is again able to communicate the ID to the All Seeing Eye making the user’s profile available to others as well. As already discussed, this kind of method “leaves some traces” namely it opens a web browser, however the corresponding malicious web page can hide its traces with a simple redirection.

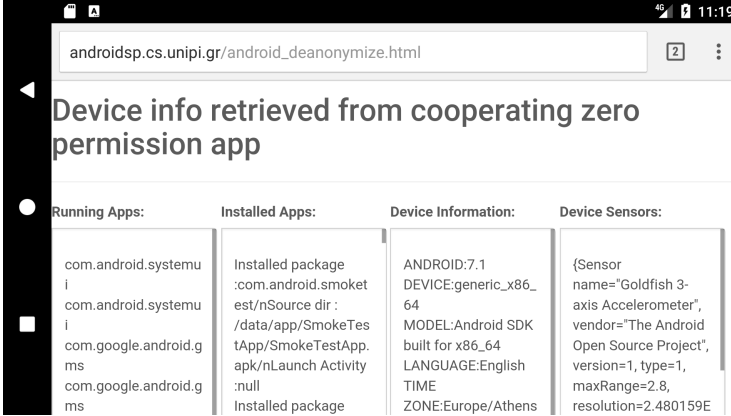
A quite significant detail in the described process is that the reached by the app web page should initially use a client side programming language in order to retrieve the transmitted ID. This is essential in order to create the local covert channel between the app and the web page, since an app ID directly transmitted to a web server would lose the ability to “find its way back” to the corresponding device’s web page.

5 Experimental results and statistics

In order to provide an estimation of the potential exposure of users to these threats, the authors of this paper used data by utilizing “Tacyt” (<https://tacyt.elevenpaths.com>). Tacyt is an innovative cyber intelligence tool that facilitates research in Android mobile apps environments with big data technology. The aim of Tacyt is to enable quick detection, discovery and analysis of these threats to reduce their potential impact on organizations. Enabling app data mining and detection, enables research and analysis of the collected information from Google Play and other markets. Due to the implementation of Tacyt, the responses are per app version, and not per app, nonetheless they provide a very good overview of apps dating back to at least three years ago.

Table 2 presents the results from the performed queries. In our first query we tried to identify how many apps provide a deep or app link. This information is declared in the manifest of each application and is clearly marked with the `android.intent.category.BROWSABLE` tag in the XML. The next two rows involve app versions which required the `ACCESS_COARSE_LOCATION` and Internet permission which could be potentially used to deanonymize users. Similarly, using PublicWWW, a source code search engine for web pages, we found more than 96,000 web pages to use geolocation features in their code for locating users.

Finally, we have implemented a proof of concept app that is able to cooperate with web pages without requesting any dangerous permissions. The app is available at androidsp.cs.unipi.gr/android_deanonymize.html. Once installed, the app recovers a lot of information from the device, such as installed and running apps, device info as well as measurements from many sensors which do not require any dangerous permissions. Eventually, as illustrated in Figure 3, when the user visits the web page through his mobile browser he can verify that the web page has recovered all this information without requesting his consent. It is worth to be noted, that apart from providing access to sensors that a web page would not normally have, the measurements for these sensors are also listed in a fine grained mode, e.g. access to accelerometer detailed measurements which could be used for fingerprinting [9].



Running Apps:	Installed Apps:	Device Information:	Device Sensors:
com.android.systemu i com.android.systemu i com.google.android.g ms com.google.android.g ms	Installed package :com.android.smoket est/nSource dir : /data/app/SmokeTes tApp/SmokeTestApp. apk/nLaunch Activity :null Installed package	ANDROID:7.1 DEVICE:generic_x86_ 64 MODEL:Android SDK built for x86_64 LANGUAGE:English TIME ZONE:Europe/Athens	{Sensor name="Goldfish 3- axis Accelerometer", vendor="The Android Open Source Project", version=1, type=1, maxRange=2.8, resolution=2.480159E

Fig. 3: Device info as obtained from a web page through a zero permission app installed in a device running Android 7.1.1.

6 Conclusions

The ever increasing use of mobile devices exposes user privacy in numerous ways. Despite the fact that mobile OSes take several measures to protect their users, attackers seem to always be one step ahead. Nonetheless, most would agree

	Google Play		Other markets	
	Available	Unavailable	Available	Unavailable
App & Deep links	432,204	87,483	71,686	34
ACCESS_COARSE_LOCATION	996,326	215,335	154,749	29
INTERNET	4,044,922	1,046,310	533,492	149
Total versions in market	4,207,542	1,095,398	576,204	155

Table 2: Results from Tacyt.

that the state of the art countermeasures guarantee an independence between the browser and the mobile apps in a way that they cannot exchange information. Taking Android as our reference platform, we introduce new methods that exploit various inherent mechanisms to practically guarantee absolute identification with limited resource usage. Moreover, the proposed methods extend the notion of device fingerprinting to what we call session fingerprinting. Our techniques can be performed without accessing unique device characteristics or using dangerous permissions. In this regard, our techniques imply a bigger threat, as the covert channel that is created between the web pages and the apps cannot be traced easily.

Due to the fact that all the aforementioned mechanisms are inherent in Android, one cannot rule out the possibility of these mechanisms already been exploited, enabling unauthorised and unregulated cooperation between the two ecosystems. Clearly, this would greatly expose users' privacy, bypassing the permission model of the most widely used mobile platform. Addressing such issues is a rather challenging task, because, apart from changing the native Android mechanisms, it is also required for the OS to determine the context of some calls, either to prohibit access to resources, or to obfuscate the underlying information, since the calls seem legitimate.

Acknowledgments

This work was supported by the European Commission under the Horizon 2020 Programme (H2020), as part of the *OPERANDO* project (Grant Agreement no. 653704) and is based upon work from COST Action *CRYPTACUS*, supported by COST (European Cooperation in Science and Technology). The authors would like to thank *ElevenPaths* for their valuable feedback and providing them access to Tacyt.

References

1. Android developers: Getting the last known location. <https://developer.android.com/training/location/retrieve-current.html> (2017)
2. Arp, D., Quiring, E., Wressnegger, C., Rieck, K.: Privacy threats through ultrasonic side channels on mobile devices. In: 2nd IEEE European Symposium on Security and Privacy (EuroS&P) (2017)

3. Beltran, V., Bertin, E., Crespi, N.: User identity for webrtc services: A matter of trust. *IEEE Internet Computing* 18(6), 18–25 (2014)
4. Bergkvist, A., Burnett, D.C., Jennings, C., Narayanan, A., Aboba, B.: WebRTC 1.0: Real-time communication between browsers. <https://www.w3.org/TR/webrtc/> (2016)
5. Blog, A.D.: Changes to device identifiers in android o. <https://android-developers.googleblog.com/2017/04/changes-to-device-identifiers-in.html> (2017)
6. Bojinov, H., Michalevsky, Y., Nakibly, G., Boneh, D.: Mobile device identification via sensor fingerprinting. *arXiv preprint arXiv:1408.1416* (2014)
7. Cáceres, M., Jiménez Moreno, F., Grigorik, I.: Network information API. <http://wicg.github.io/netinfo/> (2017)
8. Chandra, S., Lin, Z., Kundu, A., Khan, L.: Towards a systematic study of the covert channel attacks in smartphones. In: *International Conference on Security and Privacy in Communication Systems*. pp. 427–435. Springer (2014)
9. Dey, S., Roy, N., Xu, W., Choudhury, R.R., Nelakuditi, S.: Accelprint: Imperfections of accelerometers make smartphones trackable. In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)* (2014)
10. Eckersley, P.: How unique is your web browser? In: *International Symposium on Privacy Enhancing Technologies Symposium*. pp. 1–18. Springer (2010)
11. Gasior, W., Yang, L.: Exploring covert channel in android platform. In: *Cyber Security (CyberSecurity), 2012 International Conference on*. pp. 173–177. IEEE (2012)
12. Goodin, D.: Beware of ads that use inaudible sound to link your phone, tv, tablet, and pc. <http://arstechnica.com/tech-policy/2015/11/beware-of-ads-that-use-inaudible-sound-to-link-your-phone-tv-tablet-and-pc/> (2015)
13. Grace, M.C., Zhou, W., Jiang, X., Sadeghi, A.R.: Unsafe exposure analysis of mobile in-app advertisements. In: *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks*. pp. 101–112. WISEC '12, ACM (2012)
14. Kapravelos, A., Grier, C., Chachra, N., Kruegel, C., Vigna, G., Paxson, V.: Hulk: Eliciting malicious behavior in browser extensions. In: *USENIX Security*. pp. 641–654 (2014)
15. Mowery, K., Bogenreif, D., Yilek, S., Shacham, H.: Fingerprinting information in javascript implementations. *Proceedings of W2SP 2*, 180–193 (2011)
16. Mowery, K., Shacham, H.: Pixel perfect: Fingerprinting canvas in html5 pp. 1–12 (2012)
17. Popescu, A.: Geolocation api specification 2nd edition. <https://www.w3.org/TR/geolocation-API/> (2016)
18. Rushanan, M., Russell, D., Rubin, A.D.: Malloryworker: Stealthy computation and covert channels using web workers. In: *International Workshop on Security and Trust Management*. pp. 196–211. Springer (2016)
19. Stevens, R., Gibler, C., Crussell, J., Erickson, J., Chen, H.: Investigating user privacy in android ad libraries. In: *Proceedings of the 2012 Workshop on Mobile Security Technologies (MoST)* (2012)
20. Zhou, Z., Diao, W., Liu, X., Zhang, K.: Acoustic fingerprinting revisited: Generate stable device id stealthily with inaudible sound. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. pp. 429–440. ACM (2014)