Article

A Multi-Tier Security Analysis of Official Car Management Apps for Android

Efstratios Chatzoglou [1], Georgios Kambourakis [2,*] and Vasileios Kouliaridis [1]

[1] Department of Information & Communication Systems Engineering, University of the Aegean, 813 00 Samos, Greece; efchatzoglou@gmail.com (E.C.); bkouliaridis@aegean.gr (V.K.)
[2] European Union, Joint Research Centre, 21027 Ispra, Italy
* Correspondence: georgios.kampourakis@ec.europa.eu or gkamb@aegean.gr; Tel.: +39-0332-78-5013

Abstract: Using automotive smartphone applications (apps) provided by car manufacturers may offer numerous advantages to the vehicle owner, including improved safety, fuel efficiency, anytime monitoring of vehicle data, and timely over-the-air delivery of software updates. On the other hand, the continuous tracking of the vehicle data by such apps may also pose a risk to the car owner, if, say, sensitive pieces of information are leaked to third parties or the app is vulnerable to attacks. This work contributes the first to our knowledge full-fledged security assessment of all the official single-vehicle management apps offered by major car manufacturers who operate in Europe. The apps are scrutinised statically with the purpose of not only identifying surfeits, say, in terms of the permissions requested, but also from a vulnerability assessment viewpoint. On top of that, we run each app to identify possible weak security practices in the owner-to-app registration process. The results reveal a multitude of issues, ranging from an over-claim of sensitive permissions and the use of possibly privacy-invasive API calls, to numerous potentially exploitable CWE and CVE-identified weaknesses and vulnerabilities, the, in some cases, excessive employment of third-party trackers, and a number of other flaws related to the use of third-party software libraries, unsanitised input, and weak user password policies, to mention just a few.

Keywords: smart cars; digital automotive services; security; privacy; Android; vulnerability assessment; dynamic analysis; static analysis

check for updates

Citation: Chatzoglou, E.; Kambourakis, G.; Kouliaridis, V. A multi-tier security analysis of official car management apps for Android. *Future Internet* **2021**, *13*, 58. https://doi.org/10.3390/fi13030058

Academic Editor: Dino Giuli

Received: 29 January 2021
Accepted: 20 February 2021
Published: 25 February 2021

1. Introduction

Nowadays, cars get increasingly smarter and have already become part of the Internet of Things (IoT). According to Statista [1], during the last decade, there is a significant augmentation in the sales of cars with embedded telematics, while the relevant market is estimated to reach about USD 166 billion by 2025, after recovering from the adverse impact of the COVID-19 pandemic. In this context, the term "connected cars" refers to cars which are connected bidirectionally to one or more external networks in some way. Naturally, this functionality provides one with the ability to manage and even, to some extent, control cars remotely, say, by using an app on their smartphone. There exists a wide variety of apps for connected cars, which generally can be split into single-vehicle and fleet use. The focus of this work is on official single-vehicle management apps. Amongst others, such an app continuously gathers vehicle usage and service data for helping the owner to keep up with the vehicle's status in real-time. For instance, the user can be informed about the fuel and oil levels, the estimated driving range, the tire pressure, monitor the distance, fuel consumption and driving efficiency per route, lock and unlock the vehicle remotely, keep an eye on maintenance needs and schedule a service appointment, notified when remote (over-the-air) software updates are available for download, locate a parked vehicle, and many others.

On the other hand, since no standardization or software development best practices regarding the building, vetting, and maintenance of such apps, including standard protocols and libraries, are in place, there is a high risk of users exposed to privacy [2–4] and security vulnerabilities [5]. Actually, recent consumer and industry surveys have examined the security and privacy challenges related to the connected cars ecosystem [6,7]. Specifically, Ref. [6] showed that approximately 59% of European consumers are aware that their vehicle can collect an array of potentially sensitive data, and 80% would only ponder sharing such data if certain incentives are stipulated, including less expensive car insurance rates. Additionally, due to data privacy concerns, about 77% of the respondents have not used a smartphone app or service. The industry survey in [7], conducted among more than 100 global automotive leaders, pinpointed that cybersecurity and data privacy are indeed a key concern in the automotive industry. Among others, a prominent inference is that about 84% of the respondents worried about the current cybersecurity practices and considered them to be rather outdated.

Our contribution: Motivated by these facts, the work at hand is the first to our knowledge to attempt to shed light on the inner workings of this kind of official apps by analysing them under multiple diverse viewpoints. Precisely, this study contributes in revealing and pinpointing a range of significant weaknesses and security shortcomings possibly existing in this type of apps with the purpose of stimulating proper secure and privacy-preserving software engineering practices in regard to the app's whole life-cycle. Put simply, we empirically demonstrate that, as a proactive approach, security and privacy by design is of need in this rapidly ecosystem too. To this end, we employ both static and dynamic analysis methods, allowing us to dig deep and divulge multifarious security and privacy issues, which may directly or indirectly affect the end-user. Amongst them, one can discern (a) the use of an array of potentially dangerous permissions and privacy-invasive API calls, (b) several CWE- and CVE-identified weaknesses and vulnerabilities, (c) the use of third-party trackers, (d) flaws attributed to the employment of shared libraries, (e) possible user or other kinds of input sanitization hiccups, and (f) a number of security issues pertaining to the user-to-app registration and interaction.

The remainder of this paper is organised in the following manner. The next section discusses the related work. Section 3 elaborates on the methodology used to collect and analyse the apps. A first-tier static analysis focusing on apps' permissions and API calls is given in Section 4. A deeper static analysis targeting on a range of possible vulnerabilities is delivered in Section 5. The penultimate section details on the results of dynamic analysis, while Section 7 concludes and gives pointers to future research.

2. Related Work

Thus, far, considerable work has been devoted to addressing security and privacy in vehicular network systems for intelligent transportation system (ITS) usages [8]. Nevertheless, while in this ecosystem, connected car apps are a popular subject, little research has been conducted so far towards evaluating them under the security and privacy prisms. Mandal et al. [9] presented a static analysis approach to discover software vulnerabilities in Android auto infotainment apps [10]. They examined more than 20 infotainment apps available in Google Play at that time, and concluded that nearly 80% of them were potentially vulnerable. Panarotto et al. [11] examined the OpenXC library (http://openxcplatform.com/ (accessed on 20 February 2021)), which provides Android apps with a way, i.e., API, to interact with the car's hardware, and showed how this library can be exploited in the context of injection attacks. Furthermore, the authors proposed a static analysis approach which, according to the authors, nips such attacks in the bud. Recently, Wen et al. [12] proposed a cost-effective and automatic, i.e., no human intervention required, system called CANHUNTER, for reverse engineering of CAN bus commands using just connected car apps. These apps fundamentally rely on the CAN bus commands to engage with the vehicle and achieve compatibility with existing in-vehicle systems. The authors evaluated CANHUNTER by testing it on more than 200 on-board diagnostics (OBD) dongle and in-vehicle infotainment car apps acquired from both the official iOS and Android app markets.

Furthermore, when focusing on the regulation and standardization activities [13], one realises that no progress has been made so far on shaping commonly accepted methods and frameworks, including software libraries, APIs, and possibly common criteria (as that of ISO/IEC 15408), that can be used for the sake of regulating the development and maintenance of such automotive apps. For instance, while European Union regulations, including EU 2017/1151 and 2018/1832, address key issues regarding vehicle tampering, and regulate access to vehicle security features from a rather high-level standpoint, yet, they do not touch upon connected cars apps. The same stands true for the under development ISO/SAE DIS 21434 standard titled "Road vehicles—Cybersecurity engineering", which aims at ensuring that original equipment manufacturers (OEMs) and all the involved parties in the supply chain have established structured processes that endorse a security-by-design process.

Recently, the Task Force on Cyber Security and OTA issues of the Working Party on Automated/Autonomous and Connected Vehicles (GRVA) of the United Nations Economic Commission for Europe (UNECE), has drafted on Feb. 2020 two important United Nations proposed regulations and one amendment regarding the cybersecurity of the vehicles [14]. The proposal titled "Proposal for a new UN Regulation on uniform provisions concerning the approval of vehicles with regard to cyber security and of their cybersecurity management systems" [15] along with its "Proposal for the 01 series of amendments" [16], proposes certain provisions "for the approval of cybersecurity management systems as well as of vehicles with regard to cyber security". Especially, the Annex 5 of the document titled "Proposal for amendments to ECE/TRANS/WP.29/GRVA/2020/3" [17], sets out a detailed and full-fledged list of threats, vulnerabilities or attack methods related to the threats, and possible mitigation strategies. Concerning mitigation to the various identified threats, the proposal addresses the following categories: "Vehicle communication channels", "update process", "unintended human actions facilitating a cyber attack", "external connectivity and connections", "potential targets of, or motivations for, an attack", "potential vulnerabilities that could be exploited if not sufficiently protected or hardened", "data loss/data breach from vehicle", and "mitigations to the threats outside of vehicles".

Furthermore, the "proposal for a new UN Regulation on uniform provisions concerning the approval of vehicles with regard to software update processes and of software update management systems" [18], amongst others tackles issues regarding certificate of compliance for vehicle software update management, covers general specifications, including "requirements for the software update management system of the vehicle manufacturer" and "requirements for the vehicle type", refers to matters pertaining to the "modification and extension of the vehicle type" and the "conformity of production", and considers potential "penalties for non-conformity of production". All the previous proposals have been consolidated and approved last June as "UN Regulation No. 155 on Cyber Security and Cyber Security Management Systems" and "UN Regulation No. 156 on Software Updates and Software Updates Management Systems" by the World Forum for Harmonization of Vehicle Regulations, entered into force on late January 2021 [19]. Basically, both these regulations extend type approval of vehicles to lifetime management of cybersecurity and software. While all these regulation-oriented initiatives and advancements aiming at reinforcing the cybersecurity of vehicles are definitely on the positive side, as stems from the above analysis, the topic of connected cars apps is not tackled directly, but indirectly as potentially being part of other more generic categories of cybersecurity issues, say, "external connectivity and connections" and "potential vulnerabilities that could be exploited if not sufficiently protected or hardened".

3. Methodology

Thirty one apps were gathered from the Google Play Store with data freeze as of 13 January 2021. Recall that we only include single-vehicle management apps that target the European market. App details are collected in Table 1 following an alphabetical order, first per automotive group and then per app name in that group. The same sorting order is used for the rest of the tables and figures across all the sections of this work.

As shown in Figure 1, two axes of analysis were followed. The first, scrutinises each app statically, while the second examines the app by manually running it. Specifically, the static axis incorporates several stages of analysis, including sensitive permissions and API calls, and third-party trackers, which target mainly the privacy of the end-user, and misconfigurations, weaknesses and vulnerabilities, which focus on the security of the app. For instance, the latter stage looks into the code of each app for possibly identifying CWEs, CVEs, misconfigurations attributed to the use of shared libraries, etc. This stage also includes a basic taint analysis.

Static analysis employed three different tools. Two of them, namely Androtomist [20] and MobSF [21] are open-source, while the other, namely Ostorlab [22], utilised only for outdated software component analysis and taint analysis, is a software-as-a-service (SaaS) product. Details on these tools are given in the respective sections. When looking for weaknesses and vulnerabilities, we also relied on the methodology set out in the OWASP mobile security testing guide [23].

On the other hand, the dynamic analysis axis concentrates on any kind of possibly weak or misconfigured feature that pertain to the user-to-app registration phase, namely the one a new user installs and runs the app with the aim of creating a user account and registering their vehicle. As already mentioned, this type of analysis has been carried out by hand.

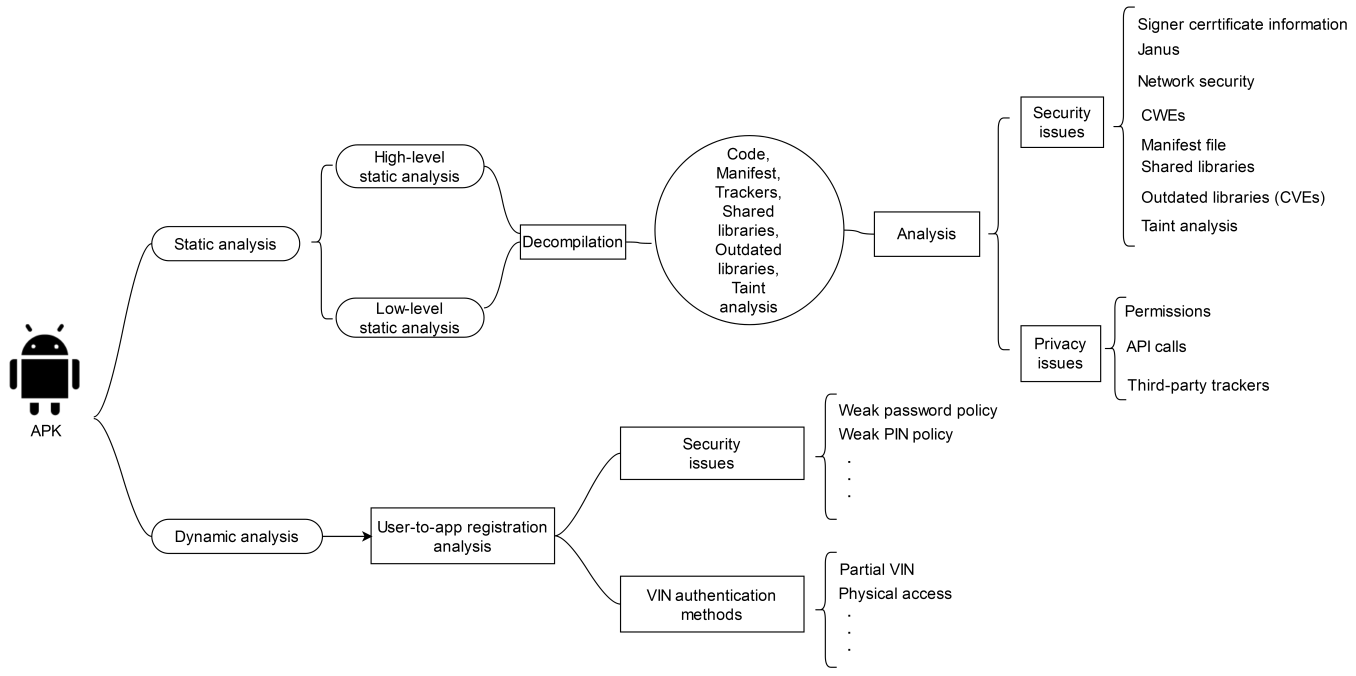

Figure 1. Overview of app analysis methods.

Table 1. Official car management apps found in Google Play Store. * In the following, they are referred to as MITSUBISHI RC and OUTLANDER PHEV RC, respectively.

Manufacturer	**App Name**	**Group**	**Examined App ver.**
Volvo	Volvo On Call [24]	AB Volvo	4.6.16
BMW	BMW Connected [25]	BMW	6.5.0.6411
BMW	My BMW [26]	BMW	1.2.2
Mini Cooper	MINI Connected [27]	BMW	6.5.0.6411
Mini Cooper	MINI [28]	BMW	1.2.1
Mercedes-Benz	Mercedes me [29]	Daimler	1.5.0
Alfa Romeo	My Alfa Connect [30]	FCA	1.10.2
Fiat	FIAT [31]	FCA	1.10.2
Jeep	My Uconnect [32]	FCA	1.13.3
FCA	UConnect LIVE [33]	FCA	2.2.17
Ferrari	MyFerrari [34]	Ferrari	1.10.2
Ford	FordPass [35]	Ford	3.14.0
Mitsubishi	MITSUBISHI Remote Control * [36]	Mitsubishi	1.0.0
Mitsubishi	OUTLANDER PHEV Remote Ctrl * [37]	Mitsubishi	3.1.1
Nissan	NissanConnect [38]	Nissan	2.1.2
Citroën	My Citroën [39]	PSA	1.26.1
DS	MyDS [40]	PSA	1.26.2
Opel	myOpel [41]	PSA	1.26.1
Peugeot	MYPEUGEOT [42]	PSA	1.26.2
Jaguar	Jaguar InControl [43]	Tata	1.81
Land Rover	Land Rover InControl [44]	Tata	1.81
Tesla	Tesla [45]	Tesla	3.10.9-433
Toyota	MyT [46]	Toyota	3.16.1
Audi	myAudi [47]	VW	3.10.1
Bentley	My Bentley [48]	VW	4.0.1
Lamborghini	Lamborghini Unica [49]	VW	4.2.21
Porsche	Porsche Connect [50]	VW	3.7
Seat	SEAT CONNECT [51]	VW	1.1.29
Skoda	MyŠKODA [52]	VW	3.7.3
Skoda	ŠKODA Connect LITE [53]	VW	2.1.9
Volkswagen	We Connect [54]	VW	5.5.3

4. High-Level Static Analysis

This section outlines all noteworthy results per app in regard to a first-tier, coarse static analysis. Extracting the security permissions listed in the AndroidManifest.xml file of an app is a key step towards understanding its overall behavior [55]. Additionally, the lookup for potentially privacy-invasive API calls in the app's code can on the one hand provide supplementary information about higher-risk actions the app may perform, and on the other, reveal whether the identified calls coincide with the requested permissions. For this purpose, as already pointed out, the Androtomist tool [20] has been employed. Specifically, for the needs of this study, the tool collected the permissions from the app's manifest file and the API calls from the smali files. Table 2 gathers the potentially privacy-invasive, "dangerous" according to the API [56], permissions requested by each examined app. Bear in mind that, in contrast to a "normal" permission, every higher-risk permission requires prompting the user. In Table 2, the following 12 permissions are identified.

- P1: READ_CALENDAR allows an app to read the user's calendar data.
- P2: WRITE_CALENDAR permits an app to write the user's calendar data.
- P3: CAMERA grants access to the camera.
- P4: READ_CONTACTS allows the app to read the user's contacts data.
- P5: WRITE_CONTACTS enables the app to write the user's contacts data.
- P6: GET_ACCOUNTS allows access to the list of accounts in the Accounts Service, namely it offers access to the existing accounts on the user's device.
- P7: ACCESS_FINE_LOCATION. This permission allows the app to access the precise location of the device via the use of GPS, WiFi, and mobile cell data. It is also required

for some connectivity tasks, including connecting to nearby devices over Bluetooth Low Energy (BLE).

- P8: ACCESS_COARSE_LOCATION is potentially privacy-invasive as it allows the app to access the approximate location of the device through the use of either or both WiFi and mobile cell data.
- P9: READ_PHONE_STATE. This permission allows read-only access to phone state. This includes the current cellular network information, the status of any ongoing calls, and a list of any *PhoneAccounts*, i.e., apps which can place or receive a phone call, registered on the device.
- P10: CALL_PHONE allows an app to initiate a phone call without going through the dialer user interface for the user to confirm the call.
- P11: READ_EXTERNAL_STORAGE allows an app to read from external storage, such as an SD card.
- P12: WRITE_EXTERNAL_STORAGE permits an app to write to external storage.

As seen from Table 2, all apps requested at least two dangerous permissions. Specifically, a total of 10, 9, 8, and 7 such permissions were asked by 3, 5, 6, and 5 apps, respectively, while only one app requested the least number of two permissions of this kind. Not less important, all apps requested the ACCESS_FINE_LOCATION permission (P7), and the vast majority of them permission to read and write the external storage (P11 and P12). Moreover, from the same table, it can be inferred that apps of the same automotive group may present an identical or nearly identical distribution of permissions. For instance, this observation stands true for the PSA, and partially for the Mitsubishi and Tata groups. Furthermore, the same allocation of permissions may be perceived for certain apps within a group, e.g., the two “connected” apps in the BMW group.

Finally yet importantly, some of these permissions, namely P2, P3, P4, P5, P6, P7, P10, and even P11 and P12 do not seem necessary for a connected cars app of this kind to possess. For instance, after interacting with the apps, we realised that several of them asked for the CAMERA permission just for scanning the Vehicle Identification Number (VIN). While this may come in handy while entering the VIN in the app, it is only to be used once or, at best, a limited number of times. Furthermore, as mentioned in the introduction, some apps may enable the user to schedule a service appointment. In this case, granting, say, P2, P5, and P10 may seem reasonable, although this utility will be used a few times per year. At the same time, however, this also magnifies the app’s attack surface, so a careful balance between functionality and usability should be maintained. Moreover, some apps may offer the possibility to view past routes on to a map or locate a parked vehicle. However, at least for the first case, choosing to grant the app with only P8, which caters for an accuracy of approximately equivalent to a city block, is way more privacy-preserving. P11 and P12 on the other hand seem justifiable if the user wishes to store historical vehicle data to a SD card. However, is this something the average user would likely do? Overall, as a rule of thumb, the number of permissions that an app requests should be minimised. This is because curtailing access to sensitive permissions diminishes the risk of unintentionally misusing some of them, and also significantly contributes in reducing the app’s attack surface. So, while such a permission may be deemed necessary for introducing a new functionality, it may compromise user adoption and render the app vulnerable for attackers.

Table 2. Dangerous, according to the API, permissions requested by each app.

App Name	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**	**P10**	**P11**	**P12**	**Total**
Volvo On Call	+	+	+	+	−	+	+	+	+	−	+	+	10
BMW Connected	+	−	+	+	+	+	+	−	+	−	+	+	9
My BMW	+	+	+	−	−	−	+	+	+	−	+	+	8
MINI Connected	+	−	+	+	+	+	+	−	+	−	+	+	9
MINI	+	+	+	−	−	−	+	+	+	−	+	+	8
Mercedes me	+	−	+	−	−	−	+	−	−	+	+	+	6
My Alfa Connect	−	−	−	+	−	−	+	−	+	−	+	+	5
FIAT	−	−	−	+	−	−	+	−	+	−	+	+	5
My UConnect	−	−	+	+	−	+	+	−	+	−	+	+	7
UConnect LIVE	−	−	+	−	+	+	+	+	+	+	+	+	9
MyFerrari	+	+	−	−	−	−	+	+	−	−	+	+	6
FordPass	−	−	+	−	−	−	+	+	+	−	+	+	6
MITSUBISHI RC	−	−	−	−	−	−	+	+	−	−	−	−	2
OUTLANDER PHEV RC	−	−	−	−	−	−	+	+	+	−	−	−	3
NissanConnect	−	−	+	+	−	−	+	+	+	+	−	+	7
My Citroën	+	+	+	−	−	−	+	+	+	−	+	+	8
MyDS	+	+	+	−	−	−	+	+	+	−	+	+	8
My Opel	+	+	+	−	−	−	+	+	+	−	+	+	8
MYPEUGEOT	+	+	+	−	−	−	+	+	+	−	+	+	8
Jaguar InControl	−	−	−	−	−	+	+	+	−	+	−	+	5
Land Rover InControl	−	−	−	−	−	+	+	+	−	+	+	+	6
Tesla	+	−	−	+	−	−	+	+	−	−	+	+	6
MyT	+	+	+	+	−	−	+	+	+	−	+	+	9
myAudi	−	−	+	−	−	+	+	+	+	−	+	+	7
My Bentley	+	−	−	+	−	+	+	+	−	−	+	+	7
Lamborghini Unica	+	+	+	−	−	+	+	+	−	+	+	+	9
Porsche Connect	−	−	+	−	−	−	+	+	+	−	+	+	6
SEAT CONNECT	−	−	−	−	−	+	+	+	−	−	−	+	4
MyŠKODA	+	+	+	+	−	+	+	+	+	−	+	+	10
ŠKODA Connect LITE	+	+	+	−	−	+	+	+	+	+	+	+	10
We Connect	+	−	+	+	−	+	+	−	+	−	+	−	7
Total	18	12	21	12	3	14	31	24	22	7	26	28	−

As already pointed out, API calls per app have been also extracted with the aim of identifying certain call to methods which may pose a threat to the user's privacy. Specifically, the following potentially privacy-invasive (sensitive) API calls were discovered in apps' smali code:

- *android/telephony/TelephonyManager; → getNetworkOperator()* returns the mobile country code (MCC) and mobile network code (MNC) of the current registered operator.
- *android/telephony/TelephonyManager; → getNetworkOperatorName()* returns the alphabetic name of the current registered operator.
- *[-15]android/telephony/TelephonyManager; → getLine1Number() returns the phone number string for line 1, e.g., the Mobile Station International Subscriber Directory Number (MSISDN) for a GSM phone for a specific subscription. It requires at least one of the following permissions READ_PHONE_STATE, READ_SMS, or READ_PHONE_NUMBERS.*
- *android/telephony/TelephonyManager; → getSimCountryIso()* returns the SIM provider's country code.
- *android/telephony/TelephonyManager; → getSimOperatorName()* returns the service provider name.
- *android/telephony/TelephonyManager; → getCellLocation()* returns the current location of the device and requires the ACCESS_FINE_LOCATION permission. This method was deprecated in Android v8.
- *android/location/LocationManager; → getLastKnownLocation()* gets the last known (cashed) location, if any, from the given provider. It requires either the ACCESS_COARSE _LOCATION or ACCESS_FINE_LOCATION permission.
- *android/location/LocationManager; → requestLocationUpdates()* is used to register for location updates from the given provider. It requires the same permission as the getLastKnownLocation() one.

- *android/location/Location;* → *getLatitude()* and *android/location/Location;* → *getLongitude()* API calls are used to obtain the latitude and longitude of the device, respectively.
- *android/hardware/Camera;* → *open()* is used to access a particular hardware camera. It requires the CAMERA permission.
- *android/hardware/camera2/CameraManager* is system service manager for detecting, characterizing, and connecting to CameraDevice.

The above mentioned privacy-invasive API calls have been grouped in Table 3 into three categories, namely cellular network, location, and camera.

Table 3. Categorisation of sensitive API calls discovered in the apps. * All class methods.

Relevant System	**API Calls**
Cellular Network	android/telephony/TelephonyManager; → getNetworkOperatorName() android/telephony/TelephonyManager; → getNetworkOperator() android/telephony/TelephonyManager; → getLine1Number() android/telephony/TelephonyManager; → getSimOperatorName() android/telephony/TelephonyManager; → getSimCountryIso() android/telephony/TelephonyManager; → getCellLocation()
Location	android/location/LocationManager; → getLastKnownLocation() android/location/LocationManager; → requestLocationUpdates() android/location/Location; → getLatitude() android/location/Location; → getLongitude()
Camera	android/hardware/Camera; → open() android/hardware/camera2/CameraManager; → *

It is noteworthy that API calls related to the cellular network may also expose the user's location. For example, the phone number (getLine1Number()) or SIM operator name (getSimOperatorName()) may reveal the user's country, while getCellLocation() returns the current location of the device. As shown in Table 4, API calls found in each app were cross-checked against the respective categories listed in Table 3. As observed from the former table, 21, 28, and 20 apps were found to include API calls pertaining to the cellular network, location, and camera, respectively. Interestingly, only one app includes zero API calls from any of these three categories. Altogether, similarities are perceived among apps that belong to the same group. This is clear for, say, the Mitsubishi and Tata groups, but it only partially applies to others. As displayed in Table 4, a last observation is that one app included camera and cellular network related API calls, for which the necessary permissions were not declared in its manifest file. Interestingly, however, both the missing permissions are announced for this app in its description in Play Store. Specifically, this app contains the getLine1Number() method, which requires at least one extra permission not declared in its manifest file. Therefore, according to the Android API, this call cannot be executed. The same app includes the android/hardware/Camera; → open() method, which according to the API, would not be able to carry out without also declaring the CAMERA permission.

Table 4. Potentially privacy-invasive API call groups per app. The asterisk indicates that at least one API call does not coincide with the permissions requested by this app.

App Name	Cellular Network	Location	Camera
Volvo On Call	+	+	−
BMW Connected	+	+	+
My BMW	+	−	+
MINI Connected	+	+	−
MINI	+	+	+
Mercedes me	+	+	+
My Alfa Connect	+	+	−
FIAT	+	+	−
My UConnect	+	+	+
UConnect LIVE	+	+	+
MyFerrari	−	−	−
FordPass	−	+	+
MISTUBISHI Remote Control	−	+	−
OUTLANDER PHEV RC	−	+	−
NissanConnect	+	+	+
My Citroën	+	−	+
MyDS	−	+	+
My Opel	−	+	+
MYPEUGEOT	−	+	+
Jaguar InControl	−	+	−
Land Rover InControl	−	+	−
Tesla	+ *	+	+ *
MyT	+	+	+
myAudi	+	+	+
My Bentley	+	+	−
Lamborghini Unica	+	+	+
Porsche Connect	+	+	+
SEAT CONNECT	+	+	−
MyŠKODA	+	+	+
ŠKODA Connect LITE	+	+	+
We Connect	−	+	+
Total	21	28	20

5. Low-Level Static Analysis

To dig deeper in each examined app, and for Sections 5.1–5.5, we took advantage of the well-known Mobile Security Framework (MobSF) in v3.2.4 [21]. MobSF is capable of performing both static and dynamic analysis and is used as a pen-testing, malware analysis, and security assessment framework. It is also one of the all-in-one tools recommended by the OWASP Mobile Security Testing Guide [23].

As summarised in Table 5, the focus of this part of study is on signer certificate information, APKiD, network security, code analysis aiming at divulging Common Weakness Enumerations (CWE), tracker analysis, manifest analysis, and shared library binary analysis. For all these categories, and for the sake of brevity, we mention only high severity (or high value according to the common weakness scoring system) weaknesses, intentionally omitting all low to medium value ones, which were numerous. For extracting these pieces of information, MobSF decompiles the provided Android application package (APK) using Dex to Java decompiler *(Jadx)* [57]; code de-obfuscation processes may also be applicable to this step.

On the other hand, the last two subsections of the current section are devoted to the use of outdated third-party software by the apps and on taint analysis. For both of these tasks, the Ostorlab tool was utilised. Ostorlab is a well-known software-as-a-service (SaaS) product to review the security and privacy of mobile apps. Note that for the sake of the reproducibility of the results, we used the free-to-use "community" edition of the tool. To our knowledge, the same tool has been exploited in the context of similar researches [58,59].

Table 5. Potential weaknesses and other security issues per app.

App Name	Signer Certificate Information	Janus	Network Security	Packers	CWE-250	CWE-330	CWE-276	CWE-532	CWE-312	CWE-89	CWE-327	CWE-295	CWE-749	CWE-919	CWE-780
Volvo On Call	SHA1withRSA (SHA256withRSA)	+	-	+	+	+	+	+	+	+	MD5, SHA1, AES-ECB	–	+	+	–
BMW Connected	SHA1withRSA	+	Insecure base config to permit clear text traffic	–	–	+	+	+	+	+	MD5, SHA1	–	+	–	–
My BMW	–	+	Insecure base config to permit clear text traffic –	–	+	+	+	–	+	+	MD5, SHA1	–	+	–	–
MINI Connected	SHA1withRSA	+	Insecure base config to permit clear text traffic	–	–	+	+	+	+	+	MD5, SHA1	–	–	–	–
MINI	–	+	Insecure base config to permit clear text traffic	–	–	+	+	+	+	+	MD5, SHA1	–	+	–	–
Mercedes me	SHA1withRSA (SHA256withRSA)	+	Domain is configured to trust user installed certificates	–	–	+	+	+	+	+	MD5, SHA1	–	+	–	–
My Alfa Connect	–	+	–	–	–	+	+	+	+	+	MD5, SHA1	–	+	–	–
FIAT	–	+	–	–	–	+	+	+	+	+	MD5, SHA1	–	+	–	–
My Uconnect	–	+	–	–	–	+	+	+	–	+	MD5	–	+	–	–
UConnect LIVE	SHA1withRSA	+	–	–	–	+	+	+	+	+	SHA1	+	+	–	–
MyFerrari	–	+	–	–	–	+	+	+	–	–	MD5, SHA1	–	–	+	–
FordPass	SHA1withRSA (SHA256withRSA)	+	–	–	–	+	+	+	+	+	MD5	–	+	+	–
MITSUBISHI RC	–	+	–	–	–	+	+	+	–	–	–	–	–	–	–
OUTLANDER PHEV RC	SHA1withRSA (SHA256withRSA)	+	–	–	–	+	+	+	+	–	–	–	–	–	–
Nissan Connect	–	+	–	–	–	+	+	+	+	+	MD5, SHA1, AES-ECB	–	–	–	–
My Citroën	–	+	–	–	–	+	+	+	+	+	MD5, SHA1, AES-ECB	–	+	+	–
MyDS	–	+	–	–	–	+	+	+	+	+	MD5, SHA1, AES-ECB	–	+	+	–
myOpel	–	+	–	–	–	+	+	+	+	+	MD5, SHA1, AES-ECB	–	+	+	–
MYPEUGEOT	–	+	–	–	–	+	+	+	+	+	MD5, SHA1, AES-ECB	–	+	+	–
Jaguar InControl	–	+	–	–	–	+	+	+	+	+	MD5, SHA1	–	–	–	–
Land Rover InControl	–	+	–	–	–	+	+	+	+	+	SHA1	–	–	–	–
Tesla	SHA1withRSA (SHA256withRSA)	+	–	–	–	+	+	+	+	–	MD5, SHA1	–	–	–	–
MyT	–	+	–	–	–	+	+	+	+	+	MD5, SHA1	+	–	–	–
myAudi	SHA1withRSA (SHA256withRSA)	+	–	–	–	+	+	+	+	+	MD5, SHA1	–	+	–	–
My Bentley	–	+	–	–	–	+	+	+	+	+	MD5, SHA1, AES-ECB	–	–	–	–
Lamborghini Unica	–	+	–	–	–	+	+	+	+	+	MD5, SHA1	+	+	–	–
Porsche Connect	SHA1withRSA	–	Domain config is insecurely configured to permit clear text traffic to these domains in scope	–	–	+	+	+	–	+	MD5, SHA1	–	–	–	–
SEAT CONNECT	–	+	Insecure base config to permit clear text traffic	–	–	+	+	+	+	+	MD5, SHA1	–	–	–	–
MyŠKODA	SHA1withRSA (SHA256withRSA)	+	–	–	+	+	+	+	+	+	MD5, SHA1, AES-ECB	–	–	–	–
ŠKODA Connect LITE	–	+	Insecure base config to permit clear text traffic	–	–	+	+	+	+	+	MD5, SHA1	–	+	–	+
We Connect	SHA1withRSA	+	Insecure base config to permit clear text traffic	–	–	+	+	+	+	+	MD5, SHA1, AES-ECB	–	–	–	–
Total	–	30	–	1	3	31	31	30	27	27	–	3	17	7	1

5.1. Signer Cert., APKiD, Network Security

Each APK is signed by the developer using a specific cryptographic hash function, say, SHA-1, and APK signature scheme version, say, v3. As observed from the second column of Table 5, seven apps indicate a different hash algorithm (SHA256) than the one actually used (SHA-1) to sign the app. Precisely, the algorithm in parenthesis indicates the hash algorithm declared in the manifest file, while the actual algorithm used is shown at the left. On top of this, five more apps, i.e., 12 in total, have used SHA-1. Nevertheless, NIST deprecated the use of SHA-1 and suppressed its use for digital signatures in 2011 and 2013, respectively, [60]. If the app has been signed with the use of SHA-1 (or MD5), collisions may be possible. This means that apps signed with the corresponding weak algorithm are prone to attacks, including hijacking the app with phony updates or granting permissions to a malicious app. For example, the attacker may be able to repackage the app after including malicious code in it. Then, given that the signature validates, they could phish users to install the repacked app instead of the legitimate one.

Another vulnerability, which is rooted in improper signature usage, is known as "Janus" (CVE-2017-13156) [61]. Namely, Janus can be exploited if the v1 signature scheme (JAR signing) is used along with Android v5.0 (API 21) to v7.0 (API 25). Specifically, Janus leverages on the possibility of adding extra bytes to APK and DEX files, without affecting the signature. As shown in Table 5, all but one (30) of the examined apps are vulnerable to Janus, namely they were signed under scheme v1 and support Android v6.

As presented in the forth column of Table 5, network security analysis pinpointed three high severity vulnerabilities. The first and more serious, namely "insecure base configuration to permit clear text traffic to all domains", means that the app is configured to possibly allow unencrypted network communications. The second, is similar to the previous one; the difference is that the scope of domains is narrower. The last vulnerability, titled "Domain is configured to trust user installed certificates" may allow an assailant to successfully exercise a man-in-the-middle (MitM) attack and decrypt the network traffic. It is to be noted that this triplet of issues refer to the network security configuration xml file an app may designate through a special entry <application android:networkSecurityConfig=""> in its manifest file under the <application> tag [62]. Seven apps were found to be susceptible to the first vulnerability, while the rest two apply to a single (different) app.

The Android app identifier (APKiD) [63] offers information about how an APK was built. Precisely, APKiD is used by a plethora of static analysis tools to identify packers, i.e., agents created by packing engines used to protect the software, protectors, obfuscations, etc. From them, the most interesting one is packers [64], originally made to safeguard the intellectual property of apps. However, nowadays, Android packers like Baidu, Bangcle, Ijiami, and others are used extensively by malware coders given that reverse engineering tools are typically unable to unpack and inspect concealed payloads within packed apps. As seen in Table 5, only one app was found to leverage packing mechanisms.

5.2. CWEs

This subsection succinctly details on all the high severity CWEs that are pertinent to each app. Notably, from them, CWE-89 currently belongs to the list of top 25 most dangerous software weaknesses [65], while CWE-295 and CWE-532 occupy positions 28 and 33 in the extended list, respectively.

- *CWE-250*: It is known as "Execution with unnecessary privileges". Typically, it means that the app may request root access privileges. Therefore, the app is potentially able to disable any security checks that will be performed by the Android operating system (OS), which resembles the case of having a rooted device. Three apps were found to be susceptible to this weakness.
- *CWE-330*: The "Use of insufficiently random values" vulnerability is related to the generation of predictable random values inside the app. This issue occurs if the app uses an insecure random number generator. In OWASP top 10 mobile risks

list (OWASP-10), this weakness is placed in the fifth position, namely "insufficient cryptography". Surprisingly, all the examined apps suffer from this weakness.

- *CWE-276*: This CWE, namely "Incorrect default permissions", occurs if the app is granted unneeded read/write permissions. So, any affected file can be potentially read/written from anyone. With reference to OWASP-10, this weakness is classified under M2, namely, "insecure data storage". As seen from Table 5, all apps were vulnerable to this weakness for at least one of the following reasons. The first, is related to the creation of a temp file, which may contain sensitive data. This is a major issue, since anyone can access folders that contain temp files, say, "/data/local/tmp/*". The second pertains to the fact that the app requests (read/write) access to the external storage.
- *CWE-532*: This weakness, namely, "Insertion of sensitive information into log file", emerges when a production app has enabled logging information to a file. While this feature may be helpful during the development stage of an app, it must be striped away before the app becomes publicly available. Put simply, an attacker could read these files and acquire any private information stored on them. All apps but one were vulnerable to this issue.
- *CWE-312*: It is known as "Cleartext storage of sensitive information", and is classified as M9 in OWASP-10. Naturally, when sensitive information, say, a username and/or password, are stored in cleartext form, anyone can read them. In some cases, this information may be stored inside the code of the app, e.g., in a configuration file. As observed from Table 5, only four apps were immune to this weakness.
- *CWE-89*: This extremely dangerous weakness, titled "Improper neutralization of special elements used in an SQL command ('SQL Injection')" is classified as M7 in OWASP-10. It occurs when the app does not sanitise or improperly sanitises input stemming from an upstream component, say, from a Web form for user authentication. All but four apps were found to be potentially vulnerable to this issue.
- *CWE-327*: It is referred to as "Use of a broken or risky cryptographic algorithm", and it belongs to M5 ("Insufficient Cryptography") of OWASP-10. This weakness relates to the usage of obsolete or risky encryption or hash algorithms. As seen in Table 5, all but two apps may potentially use at least one obsolete hash algorithm, namely MD5 or SHA-1, and nine of them support AES-ECB.
- *CWE-295*: This weakness titled "Improper certificate validation" is classified under M3 ("Insecure Communication") in OWASP-10. This happens when the app is configured to trust an insecure or self-signed or any kind of certificate. As already mentioned, this situation may allow assailants to instigate MitM attacks. Two of the examined apps suffer from this weakness due to an insecure implementation of TLS.
- *CWE-749*: It is known as "Exposed dangerous method or function", and it belongs to M1 ("Improper Platform Usage") of OWASP-10. This weakness can weaponise several serious vulnerabilities, which each time depend on the underlying vulnerable function. Specifically, more than half (17) of the apps were found to offer an insecure *WebView* implementation. The latter is used to display web content as part of an activity layout. In presence of this weakness, an attacker could possibly mount a MitM attack or even execute a Cross Site Scripting (XSS) injection. For more details regarding this issue, the interested reader may refer to the "WebView" section of [66].
- *CWE-919*: This weakness titled "Weaknesses in Mobile Applications" is directly related to CWE-749. Both of them tackle the same issue, but for a different matter. In our case, we observed that nearly one-quarter (7) of the examined apps have enabled the remote WebView debugging. That is, debug mode must be disabled before deploying a production application, otherwise anyone who can access an unlocked mobile device can easily obtain the app's data.
- *CWE-780*: This weakness is known as "Use of RSA Algorithm without OAEP". It means that the software employs RSA without encompassing Optimal Asymmetric Encryption Padding (OAEP), which in turn might undermine the encryption. Specifi-

cally, OAEP is typically used with RSA (RSA-OAEP) for offering resistance against adaptive chosen ciphertext attacks. As seen in Table 5, only one app is susceptible to this weakness.

Overall, the analysis yielded an unexpectedly number of apps being potentially susceptible to high value common weaknesses. For instance, the vast majority, if not all, i.e., 27 to 31 apps, were found to be prone to six of the considered weaknesses, while 17 of them to CWE-749. One can also easily discern a pretty much identical pattern of weaknesses among apps that belong to the same manufacturer group. As with many other software and hardware products, despite the fact that keeping up to date with common and impactful weaknesses aids in preventing security vulnerabilities and mitigating risk, this situation bespeaks a rather low priority on security features.

5.3. Tracker Analysis

This subsection is devoted to third-party trackers that may be utilised by each app. Specifically, MobSF uses the open source Exodus-Privacy [67] webapp to analyse any detected tracker in the app's source code. The focus is on six diverse tracker categories. The first one is "crash reporters". Such a tracker concentrates on the crashes that may occur during the normal operation of the app. Upon a crash event, they send a notification message to the developers, informing them about the respective error. The next category is referred to as "analytics". They gather all possible information regarding the usage of the app by the users, say, the time each user spent in the app, which features they used, and so on. On the other hand, the main purpose of "profiling" type of trackers is to gather multiple information regarding the user. Then, they attempt to profile the user with the aim of achieving and optimising personalised advertising. Another category of trackers is referred to as "identification". These can use the gathered information with the purpose of ultimately matching a digital (user) identity with the real person. The penultimate category of trackers is "ads". These trackers are specialised in serving personalised advertisements to the users. The last category is referred to as "location". By using location services along with the data provided by different sensors of the mobile device, the app, and subsequently the trackers, can obtain the geographical location of the user. After that, the user can be targeted with location-based ads. For details about the third-party trackers issue in the mobile ecosystem, the interested reader can refer to the works in [68–70].

App analysis revealed that 30 apps use at least Firebase or other Google analytics service as a method to measure users' engagement with them. Given that Google analytics can be considered as the price floor in third-party tracking, all eight apps which only embrace this type of trackers or no tracker at all (Porsche Connect) are reported to have zero trackers in Table 6. Nevertheless, we did not treat the same way third-party crash reporters, because it has been observed that they were used along with analytics, which monitor the user's behavior. Details on the 24 unique third-party trackers used by these apps are given in the following. Moreover, Table 6 sets out the distribution of these trackers per app.

- T1: *AltBeacon* [71] is a specification and open-source library for proximity beacon implementations. It is used to notify the app when a BLE beacon appears or disappears. Furthermore, it may allow Android devices to transmit beacons in the background. Exodus did not provide any category for this tracker. Nevertheless, through beaconing and geofencing technologies, an app can possibly acquire the location of users and target them with location-based ads [72]. So, we categorise this tracker as a location one.
- T2: *Appdynamics* [73] is a platform that enables one to monitor and analyse mobile device data. Exodus categorised this tracker as an analytics and profiling one.
- T3: *Branch* [74] is a mobile measurement and deep linking platform. According to Exodus, this tracker is categorised as analytics. However, among others, Branch collects the IP address of the device, and its fingerprint, including identity ID, hardware ID, brand, model, screen DPI and height.

Table 6. Third-party trackers per app.

App Name	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23	T24	Total
Volvo On Call	–	–	+	+	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	3
BMW Connected	–	+	–	–	+	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	3
My BMW	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–	–	1
MINI Connected	–	+	–	–	+	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	3
MINI	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–	–	1
Mercedes me	–	–	–	–	–	–	+	+	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	3
My Alfa Connect	–	+	–	–	–	–	–	–	–	–	+	+	+	+	–	–	–	+	–	–	–	–	–	–	6
FIAT	–	+	–	–	–	–	–	–	–	–	+	+	+	+	–	–	–	+	–	–	–	–	–	–	6
My Uconnect	–	+	–	–	–	–	–	–	–	–	+	+	+	+	–	–	–	+	–	+	–	–	–	–	7
UConnect LIVE	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0
MyFerrari	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0
FordPass	+	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	3
MITSUBISHI RC	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0
OUTLANDER PHEV RC	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0
NissanConnect	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	1
My Citroën	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–	–	–	–	1
MyDS	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–	–	–	–	1
myOpel	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–	–	–	–	1
MYPEUGEOT	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–	–	–	–	1
Jaguar InControl	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	1
Land Rover InControl	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	1
Tesla	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0
MyT	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–	–	–	+	2
myAudi	–	–	–	–	+	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	2
My Bentley	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0
Lamborghini Unica	–	–	–	–	–	–	–	–	–	–	–	+	+	+	+	+	+	–	–	–	–	–	–	–	6
Porsche Connect	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0
SEAT CONNECT	–	–	–	–	–	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	1
MyŠKODA	–	–	–	–	–	–	+	+	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	2
ŠKODA Connect LITE	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	+	–	–	–	–	+	–	2
We Connect	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0
Total	1	5	1	1	6	3	2	2	1	2	3	4	4	4	1	1	6	4	2	1	1	1	1	1	–

- T4: *LeanPlum* [75] is self-defined as a multi-channel customer engagement platform. Based on the Exodus output, LeanPlum is able to use messaging, mobile automation of marketing, app personalization, A/B testing (also known as split testing), and analytics. Based on this, Exodus classifies Leanplum as an analytics, location, and profiling tracker. Furthermore, the privacy policy contained in the LeanPlum website is not clear on whether it also applies to their mobile tracker or not. Namely, the provided policy mentions that they can perform data sharing, do a third-party data collection, and gather personal information.
- T5: *HockeyApp* [76] is a subsidiary of Microsoft corporation. It is mainly used for building, testing, releasing, and monitoring apps, including the reporting of crash reports in real-time. Exodus categorised this tracker as crash reporter.
- T6: *Demdex* [77] is a solution for audience management and it is part of Adobe's advertising ecosystem. According to [78], Demdex "captures behavioral data on behalf of Websites and advertisers and stores it in a 'behavioral data bank' ". Exodus categorised Demdex as an analytics tracker, noting that it can perform "cross-device identification", which is targeting users across different devices through profiling. Furthermore, it is able to apply "geotargeting and location-based targeting", using different technologies, such as GPS, beacons, etc. Lastly, it can gather "Real-time geo- and location-based targeting" data from a running app.
- T7: *Microsoft Visual Studio App Center Crashes* [79] creates an automatic report, which includes any necessary information related to an app crash. When the user re-opens the app, this report is sent to the App Center. In this respect, this tracker is categorised as crash reporter.
- T8: Based on Exodus, *Microsoft Visual Studio App Center Analytics* "collects real-time analytics that highlight users behavior. It also provides push notifications to mobile devices". This tracker is categorised as analytics.
- T9: *Jumio* [80] is an online identity and end-to-end ID identification tracker. It uses artificial intelligence technology to analyse and identify a user. To achieve this, it employs diverse methods, such as liveness detection, optical character recognition, face-based biometrics, etc.
- T10: *Urbanairship* [81] is an Android extension to the Google Analytics SDK.
- T11: Based on Exodus, *Gigya* [82] is an analytics tracker.
- T12: *Facebook Share* [83] enables the end-user to share a post through the app to the Facebook. It is categorised as an "ads" tracker. This is because the developer who possesses these information from the trackers, can provide personal ads using Facebook.
- T13: *Facebook Login* is a tracker which can provide the end-user with the Facebook login option. This tracker can be categorised as a profiling one.
- T14: Based on Exodus, *Facebook Analytics* is an analytics tracker.
- T15: *Facebook Places* is a geolocation tracker, which enables users to share their location from inside the app to the Facebook.
- T16: According to Exodus, *Twitter MoPub* [84] is an analytics and advertisement tracker. Actually, MoPub caters for a range of services, which can help a developer to publish personalised Twitter ads for targeting the right audience. Specifically, MoPub can aim specific advertisers, based on their highest offering price per ad and connect them to the publishers.
- T17: *Salesforce Marketing Cloud* [85] is a tracker destined to handle the communication step of marketing campaigns. It can manage audiences based on their mobile contacts and send targeted and personalised push notifications or email messages to the user of the mobile device. When a campaign is location-based, it can geofence and transmit beacon-based proximity marketing messages to its audience. It also tracks users engagement with the app using analytics. We categorised it as an adv and analytics one.

- T18: *Adobe Experience Cloud* [86] is an all-in-one cloud tool, which provides with a collection of solutions, regarding analytics, and advertising. We categorise this tracker as an analytics and ads one.
- T19: Based on Exodus, *Countly* [87,88] is an open-source analytics and profiling tracker.
- T20: *Splunk MINT* [89] can be categorised as a analytics and crash reporter tracker. It collects different kinds of data, targeting the performance and usage of the app.
- T21: According to Exodus, *Dynatrace* [90] is an analytics tracker.
- T22: *Batch* [91] is categorised as an analytics and profiling tracker. Among others, it uses push-notifications and in-app notifications based on the end-user's profile.
- T23: Based on Exodus, *Adjust* [92] is an analytics tracker, used primarily as a marketing tool.
- T24: *Optimizely* [93] is categorised as an analytics tracker. It can provide specialised tools regarding the marketing campaign of a vendor.

Summarising the above and with reference to Table 6, more than the two-thirds (23) of the apps were found to use at least one third-party tracker. Furthermore, the number of profiling and location trackers across all apps are significant, reaching a total of six and five different trackers, respectively. On the plus side, 19 apps utilise a limited number of trackers, i.e., one to three. Moreover, apps belonging to the same manufacturer group may present an identical distribution of trackers. For instance, the former observation stands true for the PSA and Tata groups, but only partially for the BMW, FCA, and VW ones.

5.4. Manifest Analysis

Based on coarse static analysis given in Section 4, we noted the use of several dangerous permissions from a privacy-invasive perspective. To deepen the analysis, we utilised MobSF to scrutinise the manifest file of each app, and possibly reveal any latent weaknesses. The focus here is on services, activities, and broadcast receivers. All of them, employ intents and intent-filters. Based on Android developer guide, all of the aforementioned components, but the intents, must be declared in the manifest file of an app [94].

Intents are basically message objects, which are used for either intra- or inter-app communication. They have three main usages, namely start an activity, initiate a service, and deliver a broadcast. There are two types of intents, namely explicit and implicit. The former is used to handle messages within the app, while the latter to transfer messages towards another capable app. On the other hand, an intent filter is an expression in an app's manifest file that determines the kind of intents the component would wish to obtain. In this respect, intent-filters are responsible for handling any implicit intent, e.g., broadcast receivers, and capturing system-oriented broadcast messages, where the specific broadcast message is included inside an intent. This means that for an app to receive such a broadcast message, it must declare in its manifest file a matching intent-filter.

The service app component can perform operations without needing a user interface (UI), e.g., transferring files from one app to another without involving the user. There are three types of services; foreground, which is discernible to the user, background, which is not directly perceivable by the user, and bound, which is used to bound a specific service with an app. Lastly, activities is a key component of any Android app. An activity, comprises a single, specific thing the end-user can perform, and it usually involves a UI. For example, when a user opens an app, the main activity is typically executed.

We observed that several of these three types of components did not declare a permission in the respective manifest file. That is, according to Android Developer security tips [66], when a component is declared in the manifest file is by default enabled to communicate with other apps. To curtail this functionality, the developer must declare among others a permission to this component. Next, any other app must possess the same permission for being able to communicate with this component. Note that this is a minimum security measure, meaning it is not enough to fully secure the component, but only reduce the magnitude of the issue. Therefore, the omission to at least provide the right permission to such a component is identified as a high severity weakness.

Specifically, the results obtained per app are summarised in Table 7. The left part of the table splits the components of interest in two categories, namely intent-filter on and off. The first (on) means that the app can send and also receive intents which target that specific component. When the received intent contains a malicious content, it can potentially compromise that app, e.g., bypass authentication [95]. The second (off) means that the app can potentially only send any information that will be requested from another app. Meaning that it is possible for the app to unwillingly leak sensitive information to an attacker. From Table 7, it is inferred that in either of these two categories and across all the three components, a considerable number of apps, i.e., 19, 24, 22 and 18, 17, 15, respectively, neglect to declare the appropriate permissions. While some similarities in the numbers can be observed among apps that pertain to the same automotive group, they do not seem to apply as a rule. Exception to this are the Mitsubishi and Tata groups.

Table 7. Potentially vulnerable components in the manifest file of each app.

App Name	**Intent-Filter on**			**Intent-Filter off**			**Content**	**Launch**	**Cleartext**
	Service	**Broadcast Receiver**	**Activity**	**Service**	**Broadcast Receiver**	**Activity**			
Volvo On Call	3	8	3	2	3	2	+	–	–
BMW Connected	1	9	3	2	4	0	–	–	–
My BMW	2	2	0	3	4	0	–	–	–
MINI Connected	1	7	3	2	5	0	–	–	–
MINI	2	2	0	3	4	0	–	–	–
Mercedes me	0	3	4	1	1	1	–	–	–
My Alfa Connect	1	0	2	0	0	1	–	–	–
FIAT	1	0	2	0	0	1	–	–	–
My Uconnect	1	1	0	0	0	2	–	–	–
UConnect LIVE	1	2	0	0	0	0	–	–	+
MyFerrari	1	1	0	0	0	0	–	–	–
FordPass	0	3	2	0	0	15	–	–	+
MITSUBISHI RC	0	0	0	0	0	0	–	–	–
OUTLANDER PHEV RC	0	0	0	0	0	0	–	+	–
NissanConnect	0	1	1	0	0	1	–	–	+
My Citroën	2	4	4	0	1	1	+	–	+
MyDS	1	5	4	1	1	1	+	–	+
myOpel	2	3	4	0	1	0	+	–	+
MYPEUGEOT	2	5	4	0	1	0	+	–	+
Jaguar InControl	3	3	1	2	0	2	–	–	–
Land Rover InControl	3	3	1	2	0	2	–	–	–
Tesla	0	2	1	2	0	0	–	–	–
MyT	0	1	2	1	0	1	–	–	–
myAudi	0	1	1	3	1	0	–	–	+
My Bentley	0	1	0	3	1	0	–	–	–
Lamborghini Unica	0	0	0	2	0	2	–	–	+
Porsche Connect	0	1	1	1	2	0	+	–	–
SEAT CONNECT	3	0	1	1	1	1	–	–	–
MyŠKODA	2	5	1	1	1	0	–	–	–
ŠKODA Connect LITE	0	3	1	0	2	0	–	–	–
We Connect	3	0	1	1	1	2	–	–	–

Moreover, it was observed that the apps of the PSA and Tata groups have enabled a task affinity for a specific activity in which the intent-filter was disabled. Task affinity [96] is related to an activity that will be re-parented to, using the proper attribute (*allowTaskReparenting*). Furthermore, the task affinity is connected with the main activity which will be launched first when the FLAG_ACTIVITY_NEW_TASK is called. Note that all apps have the same task affinity by default. This is a major issue because an attacker could potentially capture and read intents that are transferred between activities. For example, CVE-2020-0096, dubbed as “Standhogg 2.0”, can potentially exploit this issue in unpatched Android OS v8, 8.1, and 9.

The last three columns of Table 7 recapitulate information on a trinity of other important aspects related to the contents of the manifest file, namely, *Content*, *Launch*, and *Cleartext*. The first is related to the content provider [97], namely, an app component that interacts with a repository. Content providers are handy for apps that wish to provide data to other apps. This means that the content provider must permit other apps to access its data; if no permissions are set for a content provider component in the <provider> manifest element, any other app can access this content provider for both reading and writing. It is perceived that six apps were potentially vulnerable to this issue, neglecting to set the right permissions in their manifest file.

Launch on the other hand, refers to the Launch mode, i.e., the *launchMode* attribute in the <activity> manifest element, which determines how an activity should be launched. One app was found to specify a different Launch mode for the main activity, which should be launched with the "default" mode. Precisely, this app has set the launch mode to *singleTask/singleInstance* [98], meaning that it should be instantiated only once (singleton). Under the singleTask mode, the app can handle and control other activities, which have the standard configuration (singleTop activities). Moreover, an activity with the singleInstance launch mode is always the root activity of a task and no other activities will be created in the same task. As already mentioned, this issue is related to the task affinity as well. For more information on how an attack can take advantage of this matter, the interested reader can refer to [99].

Lastly, *cleartext*, that is, the *android:usesCleartextTraffic* flag in the <application> manifest element designates whether the app intends to use cleartext network traffic, including cleartext HTTP. As seen from Table 7, nine apps allow this functionality. Naturally, this may compromise the privacy of the end-user.

As a side note, for all the above mentioned three aspects, an alike distribution is perceived for apps classified under the same manufacturer group.

5.5. Shared Library Analysis

This stage of analysis pertains to the shared libraries (with the extension *.so*) an app may employ. These libraries are usually written in C and compiled with the Android native development kit (NDK) toolset [100]. Android exploits this logic for achieving better performance and reusing existing C libraries, without translating them to Java. Such libraries are loaded into memory at runtime. Security and privacy issues with the use of shared libraries have been already tracked down and clearly pinpointed in the Android literature [101–103].

To investigate if and to what degree this issue applies to the examined apps, we present the number of potentially vulnerable shared libraries per app in Figure 2. Again, we only consider high severity potential vulnerabilities, which pertain to four exploit mitigation techniques, namely, no-execute (NX), position-independent executable (PIE), Stack Canary, and relocation read-only (RELRO). These countermeasures, more accurately referred to as memory corruption mitigation techniques, are specific to C language, and if neglected may leave room for memory-based exploits, which inexorably migrate to the affected Android app. As seen from Figure 2, 27 out of the 31 analysed apps were found to incorporate shared libraries that overlook at least two of the above mentioned remedies. Actually, two of them have libraries that disregard all four of them, while 12 all but one. Moreover, apps belonging to certain manufacturer groups, i.e., PSA and Tata, demonstrate an identical pattern.

An OS that endorses the NX bit—a feature of the memory management unit of some CPUs—may tag specific sections of the memory as non-executable, meaning that the CPU will deny to execute any code residing in that region. If the NX bit is not set on the library, an attacker may be able to mount a buffer overflow. That is, frequently, such attacks place code in a program's data region or stack, and subsequently jump to it. However, if all writable addresses are non-executable (through the *-z noexecstack* compiler flag), such an attack is blocked. As observed from Figure 2, a couple of apps were found to incorporate libraries that allow for executable writable addresses in memory.

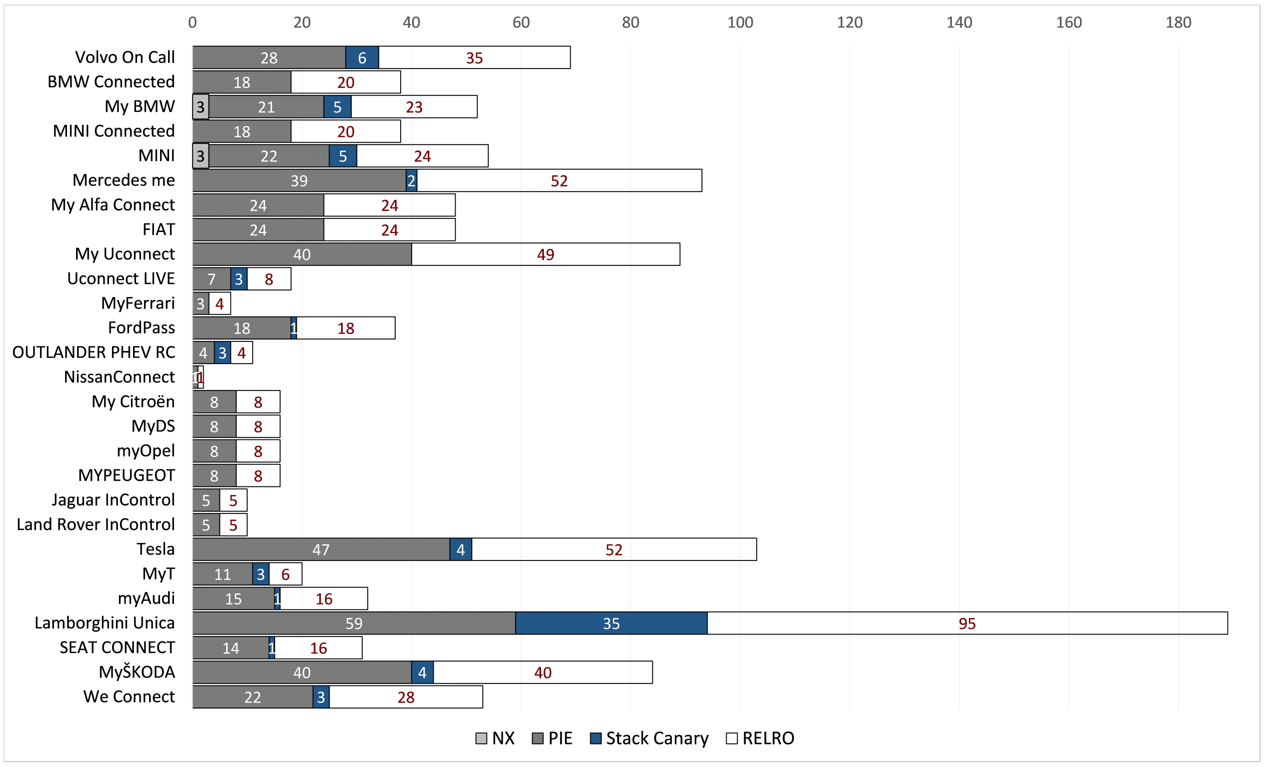

Figure 2. Number of vulnerable shared libraries per app.

PIEs on the other hand are executable binaries which are made from position-independent codes (PICs). The latter are used by shared libraries for loading their code into memory at runtime, without overlapping with other shared libraries that already exist there. Actually, this is a common mechanism to harden Executable and Linkable Format (ELF) binaries.

As seen from the same figure, 27 apps incorporate shared libraries which were built without enabling the position independent code flag (*-f PIC*). This could be exploited by an attacker, forcing the app to jump to a specific part of the memory that contains malicious code.

The use of Stack Canaries is a well-known defense against memory corruption attacks; it is a value placed on the stack with the intention to be overwritten by a stack buffer that overflows to the return address. If this protection is disabled, the app is prone to stack buffer overflow attacks, say, those that aim to overwrite certain parts of memory with malicious code. Our results show that 14 apps embrace shared libraries which neglect this defence.

Relocation Read-Only (RELRO) is another mechanism to harden ELF binaries by rendering some binary sections read-only. Precisely, RELRO ensures that the Global Offset Table (GOT)—a lookup table used by a dynamically linked ELF binary to dynamically resolve functions that are located in shared libraries—cannot be overwritten. An example of such an attack is given in [104]. With reference to Figure 2, at least one library in each app does not make use of the RELRO defence.

5.6. Outdated Software Components Analysis

For both the desktop and mobile platforms, third-party components, say, libraries, comprise one of the cornerstones of modern software development. However, as already mentioned in Section 5.5, the benefit of reusing third-party code may be largely canceled out, if that code is buggy or outdated. This may augment the attack surface of the app by far and expose end-users to security and privacy risks stemming from those external software components. Indeed, the importance of updatability of such Libraries on the Android platform has been repeatedly pinpointed in the literature [105,106]. Simply put, it has been shown that many Android apps do not update their third-party libraries, remaining vulnerable to a range of Common Vulnerabilities and Exposures (CVE).

As mentioned earlier, to look closer into this issue under the perspective of the examined apps, we employed the Ostorlab tool. The outcomes of this type of analysis per app are summarised in Table 8. As observed from the table, nearly the two-thirds of the apps (19) make use of one at least outdated library. However, as already stated, such a shortcoming is tightly connected to one or more CVEs, meaning that the respective app is susceptible to publicly disclosed security flaws. Given the plethora of the involved CVEs, in the following, we succinctly describe such issues per shared library by just enumerating the relevant CVEs. The interested reader may in addition consult the respective CVE page in [107]. For brevity, we also omit references to very well-know libraries like the *Python* one.

Table 8. Outdated third-party software components per app. The number of plus (+) signs designates how many obsolete versions of the same library exist in that app.

App Name	SQLite	OpenSSL	Python	libpng	openCV	expat	jQuery	InAppBrowser	zlib	libcurl	libjpeg	Total
Volvo On Call	–	–	–	–	–	–	–	–	–	–	–	0
BMW Connected	+	++	+	+	+	–	–	–	–	–	–	5
My BMW	+	+	+	+	+	–	–	–	–	–	–	5
MINI Connected	+	++	+	+	+	–	–	–	–	–	–	5
MINI	+	+	+	+	+	–	–	–	–	–	–	5
Mercedes Me	–	–	–	–	–	–	–	–	–	–	–	0
My Alfa Connect	+	++	–	–	–	–	–	–	–	–	–	2
Fiat	+	++	–	–	–	–	–	–	–	–	–	2
My Uconnect	+	++	–	+	–	–	–	–	+	–	–	4
Uconnect LIVE	–	+	–	–	–	–	–	–	–	–	–	1
My Ferrari	–	–	–	–	–	–	–	–	–	–	–	0
FordPass	++	+++	–	–	–	+	–	–	–	–	–	3
MITSUBISHI RC	–	–	–	–	–	–	–	–	–	–	–	0
OUTLANDER PHEV RC	–	–	–	–	–	–	–	–	–	–	–	0
NissanConnect	–	–	–	–	–	–	–	–	–	–	–	0
My Citroen	+	–	–	–	–	–	+	+	–	–	–	3
My DS	+	–	–	–	–	–	+	+	–	–	–	3
My Opel	+	–	–	–	–	–	+	+	–	–	–	3
My Peugeot	+	–	–	–	–	–	+	+	–	–	–	3
Jaguar InControl	+	+	–	–	–	–	–	–	–	–	–	2
Land Rover InControl	+	+	–	–	–	–	–	–	–	–	–	2
Tesla	–	+	–	–	–	–	–	–	–	–	+	2
MyT	+	+	+	++	+	–	–	–	+	–	–	6
My Audi	–	–	–	–	–	–	–	–	–	–	–	0
My Bentley	–	–	–	–	–	–	–	–	–	–	–	0
Lamborghini Unica	–	–	–	–	–	–	–	–	–	–	–	0
Porsche Connect	++	++	–	–	–	+	–	–	–	+	–	4
Seat CONNECT	–	–	–	–	–	–	–	–	–	–	–	0
My Skoda	–	++	–	–	–	–	–	–	–	–	–	1
We Connect	–	–	–	–	–	–	–	–	–	–	–	0
Total	16	15	5	6	5	2	4	4	2	1	1	19

Specifically, Table 8 reveals that more than half of the apps, 16 and 15, respectively, utilise at least one outdated version of the well-known *SQLite* [108] and *OpenSSL* [109] libraries. The first library, implements a lightweight, full-featured, SQL database engine, while the second provides an open-source implementation of the SSL and TLS protocols. With reference to the SQLite website (https://www.sqlite.org/cves.html (accessed on 20 February 2021)), obsolete versions of SQLite may be vulnerable to denial of service (DoS) attacks. Some pertinent CVEs to this case are CVE-2020-13632, CVE-2019-16168, and CVE-2019-19645. Even worse, several of them, say, CVE-2020-11655, CVE-2020-11656, and CVE-2019-19646 are classified as being of high or critical severity.

With respect to OpenSSL, along with DoS-related CVEs, we observed 13 others of low, medium, and high severity; CVE-2020-1968, CVE-2019-1563, CVE-2015-4000 are of low severity, CVE-2018-0737, CVE-2018-0734, CVE-2018-0735, CVE-2018-12438, CVE-2016-0703, CVE-2016-2178, CVE-2016-0800, CVE-2016-2107, CVE-2015-3195 are of medium severity, and CVE-2016-2176 is of high severity. Needless to say that these kind of vulnerabilities can be potentially weaponised for breaking the encryption between the app and the server.

Second, five apps were found to use superannuated versions of the *Python* library, namely v2.7.10 and v3.4.3. Except from four CVEs that pertain to DoS attacks, say, CVE-2018-1060, CVE-2018-1061, CVE-2017-18207, and CVE-2018-1061, we came across seven other CVEs rated as being of critical (CVE-2017-1000158), high (CVE-2017-17522), and medium (CVE-2016-0772, CVE-2016-5699, CVE-2016-1000110, CVE-2014-4616) severity. In addition, CVE-2007-4559 is not rated yet. Such vulnerabilities may lead to serious attacks, say, a TLS stripping one.

Third, six apps were vulnerable to outdated versions of *libpng* [110], which is the official platform-independent Portable Network Graphics (PNG) reference library. The vulnerable versions, i.e., v1.5.27, v1.6.14, v1.6.22beta03, and v1.6.24 suffer from three disparate CVEs, namely, CVE-2017-12652, CVE-2016-10087, and CVE-2019-7317 being of critical, high, and medium severity, respectively. Such vulnerabilities may be used as vectors to manipulate the data of a PNG image by adding malicious code. On top of that, CVE-2013-6954 can be exploited to mount a DoS attack. Note that v1.6.14 may additionally be exploited by a quartet of DoS-oriented vulnerabilities, namely, CVE-2015-8472, CVE-2015-8126, CVE-2015-0973, and CVE-2014-9495.

Five apps employed an outdated version of *openCV* [111], a library of programming functions focusing on real-time computer vision. Notably, these versions, namely v2.4.13 and v.3.3.0 are associated to more than 15 distinct CVEs. All of them are related to vulnerabilities which may cause DoS. It was observed that CVE-2017-12864 and CVE-2017-12862, both being of high severity, could potentially allow an attacker to execute code remotely.

On the other hand, four apps employed an older version (v.1.9.1) of the JavaScript *jQuery* [112] library. The aforementioned version is linked to a couple of CVEs, namely CVE-2015-9251 and CVE-2007-2379. The former refer to a XSS injection, while the latter to a JavaScript hijacking vulnerability. An equal number of apps were found vulnerable to an obsolete version (v3.0.0) of the *InAppBrowser* [113] plug-in, which provides the ability to launch a web browser within the app. Specifically, the superannuated version is linked to one critical severity CVE, namely CVE-2019-0219. This vulnerability may enable an attacker to execute arbitrary JavaScript code in the main app's webview.

Moreover, a couple of apps were found to use an obsolete version (v1.2.8) of *zlib* [114], a data compression library. This version is associated with four critical or high severity CVEs, namely CVE-2016-9840 to CVE-2016-9843, referred to as "Numeric Errors". The same number of apps were vulnerable due to the use of a superseded version (v.2.2.4) of *expat* [115], a stream-oriented XML parser library. This outdated version is prone to a couple of high severity DoS vulnerabilities, namely, CVE-2018-20843 and CVE-2019-15903. The former can be exploited by using a large number of columns, potentially causing high CPU and RAM usage, while the latter refers to a heap-based buffer overflow attack.

Lastly, a single app was using an outdated version (v7.60.0) of *libcurl* [116], a client-side URL multi-protocol file transfer library. This version is associated with three DoS-related CVEs, namely, CVE-2018-16842, CVE-2018-0500, and CVE-2018-16840. Furthermore, it is vulnerable to the high severity CVE-2019-5443, which allows for code injection. One app was found to incorporate an obsolete version (v1.5.0) of the *libjpeg* [117] library, used to read and write JPEG image files. With reference to CVE-2018-14498 and CVE-2018-20330, this version may be vulnerable to DoS attacks.

Finally yet importantly, apps that belong to the same group seem to have an identical (BMW, Mitsubishi, FCA, PSA) or approximately identical (VW) pattern regarding the use of outdated libraries.

5.7. Taint Analysis

Static taint analysis, i.e., a form of information-flow analysis, has been performed with the aid of the Ostorlab tool. Typically, this type of examination can pinpoint data leakage type of problems in the examined code. This normally refers to a variety of user or other kind of input sanitisation glitches, that may facilitate intent injection, SQL injection, password tracking, or even buffer overflows. To do so, taint analysis uses a script, which tags every private data of interest, known as the source. By following each source throughout the code as a flow, the analysis may reveal every code snippet that potentially has a leakage, the so-called sink.

After analysing all the apps, we grouped the relevant problems into three categories, namely password tracking, intent leakage, and data leakage. Regarding the first, we observed that the relevant apps used unsanitised code in their WebView login form. This issue can be exploited by an aggressor to steal any available credentials or even mount a SQL injection. On the other hand, as already mentioned in Section 5.4, intent leakage is related to three major components of the Android OS; broadcast receivers, services, and activities. Basically, properly sanitising these components in the manifest file, means less leaked intents. Lastly, data leakage has to do mainly with the usage of external storage or unsanitised data that the app receives from other apps or the system per se, and especially the WRITE_EXTERNAL_STORAGE (P12) permission, which however is employed by all but three of the apps as shown in Table 2. An important remark here is that taint analysis may produce a considerable number of false positives. Therefore, to reduce the error factor, the results of this subsection have been cross-checked with those of manifest analysis given in Section 5.4, and we kept only the intersection of the two sets based on the findings of taint analysis. Overall, an approximately 55% of the taint analysis results were found to be common between the two sets.

The results of applying taint analysis to each of the available apps are illustrated in Figure 3. The numbers inside the stacked bars designate the quantity of the issues that fall under the same category per app. As observed, the two-thirds (20) of the examined apps were found to be prone to at least one of the categories of the problems pertaining to this type of analysis. Specifically, all of these apps but three present at least one issue that is classified as data leakage, nine of them issues that belong to the intent leakage category, and only two that exhibit password tracking issues. Last but not least, a straightforward observation is that apps that belong to the same automotive group present the same issues more or less. For instance, all apps destined to the BMW group demonstrate the same distribution of issues across the same categories. The same situation applies to the PSA and TATA groups, and also partially to the VW one.

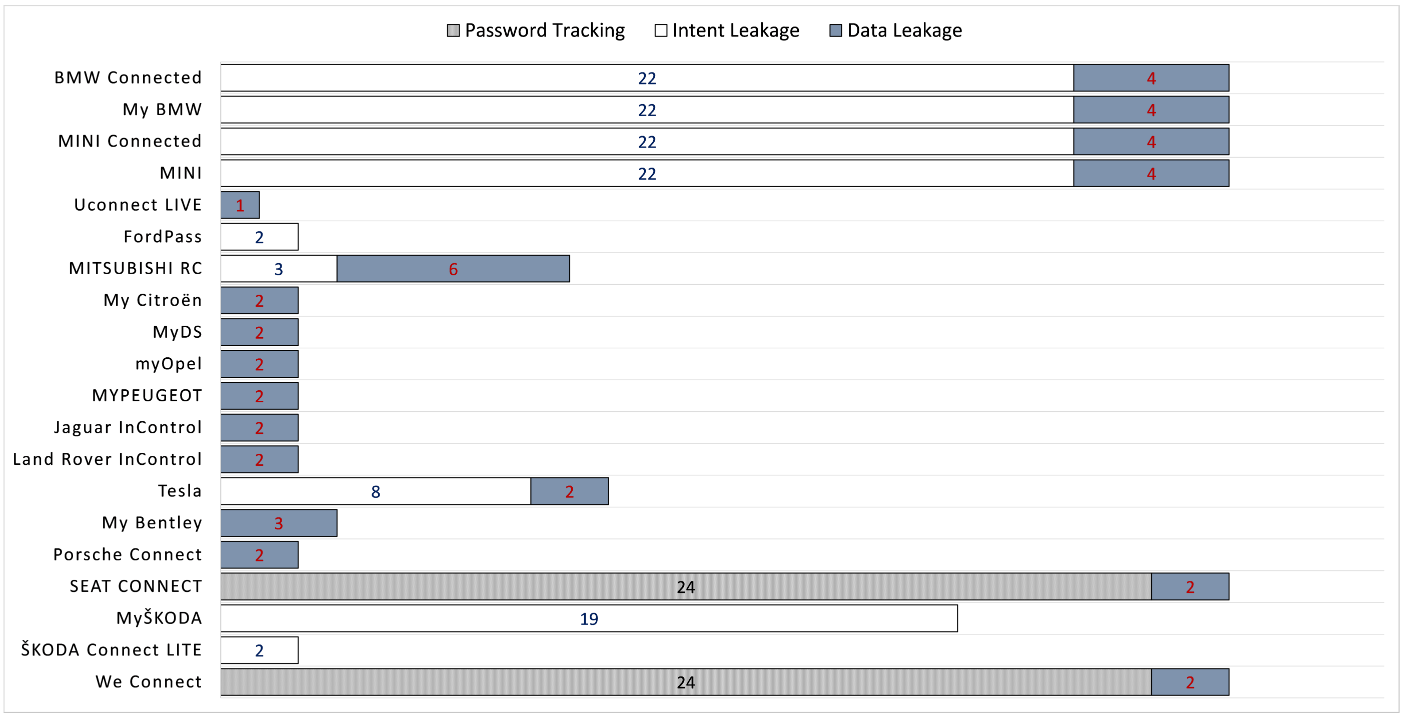

Figure 3. Issues identified through taint analysis. The numbers inside the bars designate the amount of issues per category.

6. Dynamic Analysis

6.1. App Exploration

This section details on the obtained results after dynamically testing the different apps from the perspective of the end-user, typically the car owner. Given that it is practically unworkable for one to possess or control even in the short-term so many different vehicles, this scrutiny is confined to the owner-to-app and/or owner-to-car registration (or pairing) phases and not to the app functionality in general. Namely, this phase has been carried out manually and involves the creation of a new user account, typically from inside the app or through redirection to a website. This normally requires the user to accept terms and conditions (T&C) along with other policies, if any, and enter basic identification information, including their name and surname, country, city, email address, a username and password, etc. Where applicable, it can also include registration of the vehicle against the owner, i.e., the owner is required to provide some kind of ownership-proof.

For this purpose, a Xiaomi Redmi Note 8 Pro smartphone, running on Android 10 has been employed. For the apps that required, typically after the user registration phase, a VIN as input, we tested specific vendor VINs that we either generated using a custom-made Python script or acquired from the internet by searching relevant websites [118]. In their majority, these apps requested a full 17- character VIN, except a limited number of cases where a partial VIN was needed. Where applicable, a maximum number of 25 VINs have been tested against the app.

All in all, as shown in Tables 10 and 9, this subsection compares the apps based on 20 criteria split into two categories, namely security issues and VIN authentication methods. The specific results per app are given in the following.

Table 9. Dynamic analysis security issues. S1: Acceptance of internet or randomly generated VIN, S2: Weak password policy, S3: User registration through website, S4: Weak PIN policy, S5: Informational errors to the user, S6: No password confirmation field, S7: Non-informative or improper error messages, S8: User session is kept alive after terminating the app, S9: Acceptance of disposable temporary e-mail address, S10: One-time 6-digit code.

App Name	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**
Volvo On Call	–	+	+	+	–	–	–	+	+	–
BMW Connected	+	+	–	+	–	+	–	+	+	–
My BMW	N/A	+	–	+	–	–	–	+	+	–
MINI Connected	N/A	+	–	+	+	–	+	+	+	–
MINI	N/A	+	–	+	–	–	–	+	+	–
Mercedes me	+	N/A	–	N/A	–	N/A	–	+	+	+
My Alfa Connect	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
FIAT	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
My Uconnect	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
UConnect LIVE	–	+	–	N/A	–	–	–	+	+	–
MyFerrari	N/A	–	+	N/A	–	–	–	N/A	+	–
FordPass	+	+	–	–	–	+	–	+	+	–
MITSUBISHI RC	N/A	N/A	N/A	N/A	–	N/A	–	N/A	+	N/A
OUTLANDER PHEV RC	N/A	N/A	N/A	N/A	–	N/A	–	N/A	+	N/A
NissanConnect	N/A	–	–	N/A	–	–	–	+	+	–
My Citroën	+	+	–	N/A	–	–	–	+	+	–
MyDS	N/A	+	–	N/A	–	–	–	+	+	–
myOpel	+	+	–	N/A	–	–	–	+	+	–
MYPEUGEOT	+	+	–	N/A	–	–	–	+	+	–
Jaguar InControl	N/A	+	+	N/A	–	+	–	+	+	–
Land Rover InControl	N/A	+	+	N/A	–	+	–	+	+	–
Tesla	N/A	+	+	N/A	–	+	–	+	+	–
MyT	N/A	+	–	N/A	–	–	–	+	+	–
myAudi	+	+	–	–	–	+	–	+	+	–
My Bentley	–	N/A	–	N/A	N/A	N/A	N/A	N/A	+	N/A
Lamborghini Unica	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Porsche Connect	–	+	+	N/A	–	–	–	N/A	+	–
SEAT CONNECT	–	+	–	+	–	+	–	+	+	–
MyŠKODA	+	+	–	–	+	+	+	+	+	–
ŠKODA Connect LITE	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
We Connect	–	+	–	+	+	+	+	+	+	–
Total	8	20	6	7	3	9	3	21	26	1

Table 10. Different VIN registration/authentication methods. V1: Paid subscription, V2: Limited model support, V3: Face-to-face proof of ownership, V4: Partial VIN, V5: PIN in vehicle, V6: Physical access, V7: Online proof of ownership, V8: Acceptance of unsupported model VINs, V9: Email address verification with PIN, V10: Acceptance of the same VIN with multiple accounts without verification.

App Name	**V1**	**V2**	**V3**	**V4**	**V5**	**V6**	**V7**	**V8**	**V9**	**V10**
Volvo On Call	–	–	–	–	–	–	–	N/A	+	N/A
BMW Connected	–	–	–	+	+	+	–	+	–	+
My BMW	–	–	–	–	–	–	–	–	–	–
MINI Connected	–	–	–	+	N/A	N/A	–	N/A	–	N/A
MINI	–	–	–	–	–	–	–	–	–	–
Mercedes me	–	–	+	–	–	–	–	+	–	–
My Alfa Connect	+	+	–	–	N/A	N/A	N/A	N/A	–	N/A
FIAT	+	+	–	–	N/A	+	N/A	N/A	–	N/A
My Uconnect	+	+	N/A	N/A	N/A	+	N/A	N/A	N/A	N/A
UConnect LIVE	–	–	–	–	–	+	N/A	N/A	–	N/A
MyFerrari	–	–	+	N/A	N/A	N/A	+	N/A	–	N/A
FordPass	+	–	–	–	–	–	–	+	–	+
MITSUBISHI RC	–	+	–	–	–	+	–	–	–	–
OUTLANDER PHEV RC	–	+	–	–	–	+	–	–	–	–
NissanConnect	–	+	N/A	–	N/A	N/A	N/A	–	–	N/A
My Citroën	–	–	–	–	–	–	–	+	–	+
MyDS	–	–	–	–	–	–	–	+	–	+
myOpel	–	–	–	–	–	–	–	+	–	+
MYPEUGEOT	–	–	–	–	–	–	–	+	–	+
Jaguar InControl	+	+	N/A	–	N/A	N/A	N/A	N/A	–	N/A
Land Rover InControl	+	+	N/A	+	N/A	N/A	N/A	N/A	–	N/A
Tesla	–	–	+	N/A	N/A	N/A	N/A	N/A	N/A	N/A
MyT	–	+	–	–	–	N/A	–	–	–	N/A
myAudi	–	–	+	–	–	–	–	+	–	+
My Bentley	–	+	N/A	–	N/A	N/A	N/A	N/A	–	N/A
Lamborghini Unica	+	–	N/A	–	N/A	N/A	+	N/A	–	N/A
Porsche Connect	–	–	N/A	–	N/A	N/A	+	N/A	–	N/A
SEAT CONNECT	–	–	–	–	–	+	–	N/A	–	N/A
MyŠKODA	–	–	–	–	–	+	–	–	–	N/A
ŠKODA Connect LITE	–	–	–	N/A	N/A	+	–	–	–	N/A
We Connect	–	–	–	–	–	+	–	N/A	–	N/A
Total	7	10	4	3	1	10	3	8	1	7

Volvo On Call: User registration is done through a website at https://volvoid.eu.volvocars.com/Account/initaccount?market=GB&language=en (accessed on 20 February 2021). The user has to enter an email address, password, along with a confirmation field, first and last name, country, and language. Regarding the password, the app accepted "111111wW". After registering, we managed to successfully add a vehicle with a VIN obtained from the internet. Then, the app informs the user to wait for a PIN to be sent to the email address provided. No such email was received though. The app provides a T&C and a privacy policy.

BMW Connected: It uses a distinct authentication method to connect the app with the vehicle. Specifically, after user registration, performed from inside the app, it requests the last seven characters of VIN. Then, the app informs the user that the car dashboard shows a 6-number PIN, which needs to be entered in the app. It was observed that with every wrong answer, the app directs the user back to entering the VIN; no upper bound to the number of wrong answers seems to apply. User registration requires entering the country, email, password, confirm password, first and last name, title, and a 4-digit PIN, where it accepted "1234". An 8-character password policy is enforced, but it is rather weak as the app accepted the "111111QQ" one. After user registration, the app presents the user with a privacy policy. It was noticed that the app keeps the previous session alive, so subsequent user logins are done by means of the 4-digit PIN. Another interesting point is that after entering a VIN that was already paired with another user account, the app asks the current user if they wish to change region; however, by accepting it, the app asks for vehicle 6-number PIN verification.

My BMW: First, the user is required to accept T&C. Before registering a newly created user account, the app informs the user that an account should have been already created on their behalf by their local car dealer. In the relevant registration form, the app provides a password confirmation field and accepted "1111111a" as a password. Next, an email address validation step is required. After login, the user is prompted to enter a 4-digit PIN; the app accepted the "1234" value. Next, the app requested accepting two diverse T&C and two different privacy policies and granting specific permissions, namely calendar, camera/photos, location, and notifications. We tried more than 20 VINs, but no one of them was accepted. After reopening the app, it requested either for fingerprint access or the 4-digit PIN, meaning that no password was required. This functionality applies to the "MINI Connected" app as well.

MINI Connected: User registration is performed from inside the app. Regarding password policy, it required at least a combination of eight letters, numbers, and/or symbols, accepting the "123456ab" one. Furthermore, it did not offer any option for language, automatically providing the language pertinent to the user's location. While the registration process was successful, user login failed after displaying the message "An error occurred while authenticating. Please try again." In a subsequent attempt to log in, the app requested accepting T&C and a privacy policy. After that, we were requested to provide a 4-digit PIN and validate it; the "1234" was accepted. Then, the app asked for several permissions, as given in Table 2. We tried to login/logout many times to the app. The result was to be asked every time to accept T&C and provide a PIN. To register a vehicle, the app required the last seven characters of VIN. Based on the vendor, we tried 25 different combinations, but no one of them was accepted.

MINI: It is identical to the "My BMW" one. In fact, we logged in using the BMW account credentials. Moreover, the user is required to enter a 4-digit PIN and accept T&C.

Mercedes me: User registration is performed from inside the app. First, the user needs to enter an email address, which is verified by means of a 6-digit code. Next, they have to choose a country, and provide their title, first and last name, and date of birth, but not a password. We managed to pre-register two VINs with this app. For one of them, as a next step, the app required the owner to get authorised at the premises of a local Mercedes car dealer. For the other, probably because the respective VIN corresponded to an unsupported model, the app did not request for car dealer authentication, but it did not let the user to proceed with any other operation. The app provides a T&C and a privacy policy. Given that user registration is password-less, for them to reconnect to the app, they need to provide their email, get a 6-digit code, and enter it into the app; this means that the app enforces a kind of one-time password policy.

My Alfa Connect: It requires a paid user subscription for letting them to further interact with it. That is, the owner needs to first perform an activation step at an Alfa Romeo dealership. Then, the user receives an email which contains a hyperlink, enabling them to finish registering the account and activating the relevant services on both the app and the https://www.alfaromeoconnect.eu/ (accessed on 20 February 2021) website. Given that the registration procedure requires personal interaction, it was infeasible to analyse this app further. Currently, the app supports only two 2020 models. Neither a T&C nor a privacy policy statement is provided by the app at startup.

FIAT: It is very similar to the "My Alfa Connect" one. In the app's description in Google Play, it is mentioned that "FIAT mobile app and Uconnect Services are available for FIAT vehicles equipped with the new Uconnect Box". So, as with the My Alfa Connect case, it was unattainable to proceed any further with this app.

My Uconnect: After starting the app, the user is informed that it currently supports specific models, which are equipped with the "Uconnect infotainment system". Lastly, similar to the "Uconnect LIVE" one, this app required a paid subscription.

Uconnect LIVE: This app supports FCA vehicles, including those from Abarth and Lancia. The app does not have a strong password policy; simple passwords like "12345678" are accepted. On the plus side, the app needed physical access to the vehicle for being fully enabled. Physical access can be achieved by means of USB or Bluetooth, with the preferred one being over Bluetooth. On top of that, this app requires a paid subscription.

MyFerrari: It requires a validation directly from Ferrari for authenticating the owner and connecting the app to the vehicle. Specifically, for registering, the app redirects the user to a Ferrari webpage at https://www.ferrari.com/en-US/auto/owner-reg (accessed on 20 February 2021). There, we did provide all the necessary (fake) personal information, but as a further step, the owner is required to get in contact with a local Ferrari dealer. In the user registration form, the password policy was at least 8-characters, one uppercase, one lowercase, one digit, and one special character. Another observation is that upon starting the app it asks the user to accept granting a location permission. The app provides the user with a T&C and a privacy policy.

FordPass: At start, the app asks for country information, and whether the user permits the use of cookies. However, even if the latter option is denied, the app proceeds and allows them to create an account. In this step, the user has to accept the T&C and a privacy policy. It was observed that the app accepted "12345678" as a password. Next, the user has to provide a 4-digit PIN, and as with other similar apps, the "1234" value was accepted. Note that the app does not require the user to validate their email address. After that, the user is presented with various marketing options, but the app let them to refuse all. As a safety measurement, the app also required the user to confirm that they are not driving at that moment. After completing the above mentioned steps, we were able to provide a VIN. From a total of 10 VINs, seven were accepted. For vehicle models prior to 2012, the app only displayed service maintenance data. However, for newer vehicle models, an additional authorisation step popped up. That is, the user has to self "authorise", the specific vehicle (to be controlled by her), except the vehicle is already authorised by another user. In the latter case, the user can either send a request to the already authorised user to grant her an additional authorization or deauthorise all users who have previously gain authorization to the same vehicle; we however did not try the latter option for avoiding causing any problem to legitimate users. Lastly, we noticed that a paid subscription to specific services, including traffic avoidance information can be performed from inside the app.

MITSUBISHI RC: At start, the app requires accepting T&C and a privacy policy. It also asks to be granted a couple of permissions, namely Location and Bluetooth. Then, the user needs to connect the app to the vehicle, i.e., turn on the engine and select to register from the dashboard via Bluetooth. Given the requirement of having physical access to the vehicle, we could not test the app any further. Currently, the app supports only two vehicle models.

OUTLANDER PHEV RC: It is similar to the "MITSUBISHI RC" one. The only differences are that this app is compatible with another Mitsubishi model, and the pairing process with the vehicle is done over WiFi.

NissanConnect: User registration is done from inside the app. The relevant form requires an 8-character password and includes a password confirmation field. The password "eE#11111" has been accepted. Next, the app required the user to accept T&C and validate their email address. After login, the user can register a VIN. The app itself does not inform the user about the supported models. However, according to information in Google Play, it is currently compatible with only three models manufactured from 2019 onward. Although we entered 25 VINs, no one has been accepted by the app.

My Citroën: User registration is performed from inside the app. The displayed form contains fields for entering email, password, confirm password, title, first and last name. A password of at least eight characters is required, and the app accepted the "1111111q" one. After creating a user account, the app asks for a VIN. The app did accept a VIN, and we were able to learn the periodic car maintenance data for that vehicle.

MyDS: It is very similar to the "My Citroën" one. We were able to be registered as users using the password "1234567a". The registration process also requires a VIN, but after 25 attempts, no VIN was accepted.

myOpel: It has the same graphical user interface (GUI) as that of the "My Citroën" app, and user registration is performed the same way too. After some tries, the app accepted two VINs, and we were able to see the corresponding vehicle's maintenance data, both for already completed services and future ones. The app provides the user with a T&C and a privacy policy.

MYPEUGEOT: It offers a similar GUI as with the My Citroën app. First, the app requested to accept the T&C and a privacy policy. Then, user registration required providing personal details, namely email, first and last name, password, and title. Password policy is identical to My Citroën. After email verification, one can log in to the app. We successfully registered one VIN, enabling us to see that vehicle's service maintenance data.

Jaguar InControl: Connecting the app to the vehicle requires a paid user subscription along with a pre-installed hardware component. Regarding its GUI it is almost identical to the "Land Rover Incontrol" one. After opening, the app requested to be granted a location permission. During user registration, which is done externally in an official Jaguar webpage at https://incontrol.jaguar.com/jaguar-portal-owner-web/select (accessed on 20 February 2021), the user needs to provide the country, first and last name, email, and password. The app accepted "qwertyQ1" as a password. After email verification, the user is transferred again to the webpage, where they can login. After login, they are presented with another, more detailed registration form, which also asks for a 4-digit PIN, accepting the"1234" value. After that, from inside the webpage, the user needs to add a vehicle, namely a VIN. The privacy policy, requires double acceptance, meaning ticking the corresponding boxes and also opening the documents and pressing "accept".

Land Rover InControl: It also requires a paid user subscription. During the user registration phase, the app redirects them to the Land Rover's official webpage at https://incontrol.landrover.com/jlr-portal-owner-web/request-account (accessed on 20 February 2021). The second phase is similar to the "BMW Connected" app, namely the app requested only the last eight characters of VIN. After 25 attempts, no randomly generated partial VIN was accepted. User password policy is identical to that of the "Jaguar Incontrol".

Tesla: This app offers only a login option, meaning that the user should have been already registered elsewhere. Tesla provides a registration service at https://auth.tesla.com/login (accessed on 20 February 2021), so we attempted to register through this website. The relevant form accepted the password "1234567q". After registering, we were able to login to the app, but no other option, such as registering a VIN, was enabled. It seems that Tesla pre-authenticates the owner, that is, provides each car owner with a specific account upon purchasing the vehicle.

MyT: User registration is done from inside the app. It requires providing personal information, including email address, first and last name, and a password along with its verification field. The app accepted the string "111111a@"as a password. Next, it displayed a T&C and a privacy policy. After email address validation, we were able to log in to the app and successfully entering a VIN. Nevertheless, no information has been made available for the corresponding vehicle. Instead, the app informs the user that such information will be accessible after they verify ownership against a local dealer: "This vehicle needs to be verified and only after this functionality is available". Note that when purchasing a Toyota car, typically, the car dealer will register the buyer as the owner of the car, automatically granting them access to the so-called connected services.

myAudi: It has a similar GUI to that of the MyŠKODA app. As a first step, the app asks for user's registration in terms of email address (acting as username) and password. This is done from inside the app. In this stage, the app mandated an 8-character password and accepted "11111112"; no password confirmation field were present in the form. After, the user can log in and enter their personal details, i.e., first and last name and accept the displayed marketing policy, which however is not a requirement. Next, the app asks for a VIN. After some tries, we succeeded pre-registering a couple of VINs with the app. However, to complete registration, the app requires the user to bring all the necessary documents to a local Audi dealership, who is responsible to confirm the ownership of the specific vehicle and provide them with access to the app.

My Bentley: First, the app informs the user regarding the related vehicle models it supports. Then, the user registration phase initiates from inside the app. After providing all the personal data, i.e., user location, car vendor, first and last name, email, phone number, and address, the app requested a VIN. From the 25 inputted VINs no one was accepted, probably because the supported by the app models are new. The app presents the user with a T&C and a privacy policy.

Lamborghini Unica: A user invitation code is required for them to be able to create an account and log in against the app. This code can be obtained upon user request through the app to Lamborghini. The request mandates filling in several fields, namely local dealer, VIN, and ownership documents. The app presents the user with a T&C and a privacy policy.

Porsche Connect: At start, it requires an online registration and verification phase, which is done against a website at https://login.porsche.com/auth/gb/en_GB/registration (accessed on 20 February 2021). In the relevant web form, the owner must upload all the vehicle ownership documents along with personal information, including an email address along with the corresponding verification field, first and last name, and address. The string "11111wW@" was accepted as a password. After that, the app notifies the user that they will be informed by email about the outcome. User login is not done via the app, but through the same website. After trying to log in, we redirected back to the app getting the error "There is no vehicle stored for your Porsche ID. Please register your Connect-enabled vehicle at My Porsche. (error: 9_MP_EL).". The app provides the user with a T&C and a privacy policy.

SEAT CONNECT: The app first asks for an email address. After validation, user registration is required. The app accepted "12345671" or "11111112" as a user password, but not the "12345678" one. Then a 4-digit user PIN was required; as with all other similar apps the "1234" value was accepted. We were able to pre-register one VIN with this app. Nevertheless, the app requires physical access to the vehicle for completing the user registration phase. That is, the user must possess a device named *Dataplug* [119], which is connected to the OBD port of the vehicle. The app displayed a T&C and a privacy policy.

MyŠKODA: For initiating the user registration phase, the app requested for an email address. Then, from inside the app, the user needs to enter a password; no confirmation field was present, and the app accepted "11111112". Next, the user is presented with a T&C and a privacy policy, and also needs to validate their email address. After login, the user has to accept or deny a marketing policy, which however its acceptance was mandatory for the app to proceed. Afterwards, the app displayed the error message "Your connection is not secure! Personal information may be leaked!", with error code "ERROR_CONNECTION_NOT_SECURED". After manually checking the code of this app, we realised that it creates several http connections, i.e., new *HttpURLConnection* object instances are obtained by invoking the *openConnection()* method on a *URL* object. The app accepted two VINs and informed us that one of these vehicles "is equipped with connectivity digital service technology". After that, the app requested once more the user to accept the marketing policy, and in a next step, to provide a country, language, and first and last name. Once again, the user has to accept T&C and a privacy policy, and consent to the marketing policy, which however this time can be denied. While some VINs were registered successfully, the app did not provide access to any piece of data related to these vehicles.

ŠKODA Connect LITE: It presents the same GUI as the "MyŠKODA" one. For using the app, the user must possess a dataplug. This app showed a strange behavior. After we attempted to connect our smartphone over Bluetooth with another, not of the same group, vehicle, this app started asking for authorization in the background. Precisely, the app popped up the message "Authorized workshops are being updated", but at that time, the smartphone had no internet connection of any kind, so this action can be regarded as an attempt to gain unauthorised access. By manually terminating this service after the vehicle's Bluetooth was disabled, made it to reappear, showing the same message, after approximately 5 min.

We Connect: It is similar to the "SEAT CONNECT", also requiring a 4-digit PIN. After login, we tried unsuccessfully to register a VIN, sometimes getting the error "Something went wrong. VIN backend unknown". Furthermore, as with the "SEAT CONNECT", it seems that this app requires a dataplug to the vehicle for completing the owner's registration phase.

6.2. Discussion

Based on the review done in the previous subsection, the following key observations can be drawn out. First off, with reference to Table 9, nearly one-third of the apps accepted a randomly generated or internet-acquired VIN. Even worse, for those of them that do not require any kind of ownership-proof, it was possible to see information regarding vehicle maintenance, which, although not critical, is certainly privacy-invasive. Second, for most apps, user registration does not mandate a strong password creation policy [120]; this situation stands true for almost the two-thirds of the apps. In addition, nine apps do not provide a password confirmation field, which may lock the user out of their account and subsequently force them to execute a password recovery process. However, if, say, the latter is exercised over email, then its security directly depends on the security of the email service. Interestingly, one app supports password-less user authentication via a one-time code sent over email. This option may seem more secure, but it again relies on the security of the email service, which naturally cannot be guaranteed [121]. Furthermore, the user will be unable to login to the app in absence of an internet connection or if their email service is down.

Third, a limited, but not insignificant number of apps, perform user registration externally to the app. This may seem harmless, but it is additionally always dependent on the security level of the website per se. Where a 4-digit PIN is required as a supplementary user authentication method, feeble PINs are accepted. This is of major importance because anyone who is able to snatch an unlocked phone can potentially access the app by simply entering "1234". More than two-thirds of the apps keep the active session alive even if the app is terminated by the user, without logging out. Obviously, this issue is directly associated with the use of weak PINs. Not less important, during user registration, all apps did accept a disposable temporary e-mail address, like the ones provided by Temp Mail [122]. Clearly, such a practice should not be accepted, because, among others, it is used as a method to create phony accounts or bypass areas that require a registered account. Put simply, the use of a disposable email address in the context of such an app is a clear sign of a user who is likely to engage in shady behavior.

On the other hand, regarding Table 10, which focuses on VIN-to-owner registration issues, one can infer the following. Some apps apply stricter methods to verify proof of ownership. Precisely, four of them demand in-person contact, while others in addition support on-line registration. Paid subscription is also mandated by seven apps. Secondly, three apps ask for a partial VIN, but this is not a limitation because the same apps demand additional authentication tokens, which in turn require physical access to the vehicle. Nearly one-third of the apps accept registering VINs that belong to unsupported models. Surely, this functionality is without a purpose and may yield unforeseen problems. Furthermore, a number of apps seem to accept the same VIN to be registered with different user (email) accounts, but this is done without verification. Naturally, this issue is directly linked to S1 of Table 9.

Lastly, similar to the other types of analysis, apps that belong to the same manufacturer group present the same or nearly the same characteristics. This is evident in both Tables 9 and 10.

7. Conclusions

The connected vehicle app development is a relatively new field, and the demand for such apps is expected to mushroom in the next few years. Furthermore, as the connected car landscape is still at a nascent stage, developers may be tempted toward adapting already existing apps to connected vehicles, instead of designing and engineering new ones. On top of that, the reality has proven that security and privacy aspects are often not properly prioritised by software developers and vendors who rather concentrate on the functional ones. However, as a guidepost, security must not be left anymore as an afterthought. Instead, one must proactively secure software—think for instance about security and privacy by design—while they are building it. Especially, focusing on the Android ecosystem, there are already several noteworthy standards and best practices for developing secure software, including the Android developer website, the OWASP mobile top 10 project, the CERT Android secure coding standard, and the JSSEC Android application secure design/secure coding guidebook.

In this context, the results of this work, which stem from the whole population of official single-vehicle management apps offered currently in Europe, can be used as a basis and reference for not only alerting and motivating the involved parties to proactively improve the security robustness of their products and conduct frequent vulnerability assessments, but also to raise the security awareness of end-users. From a security viewpoint, it was demonstrated that a significant number of apps remain susceptible to several high or even critical severity weakness and vulnerabilities, which are due to sundry reasons, including misconfigurations, secure coding negligence, and latent or overt security flaws that relate to the use of third-party software. Overall, at least 87% of the apps were found to be potentially exposed to six out of the total 11 CWEs, with about 80% of them to utilise a couple of obsolete cryptographic hash algorithms. Moreover, roughly 93% and 87% of apps exhibited more than one issue in their manifest file and shared libraries, respectively. At the same time, outdated software component exploration and taint analysis revealed that almost two-thirds of the apps have at least one issue. Furthermore, from a privacy standpoint, there exist several hiccups that emanate from the over-claim of sensitive permissions and API calls, the inclusion of trackers, and frail user- or vehicle-to-app registration practices. While apparently all these aspects target on increasing app's functionality and improve user experience on the app as a whole, they are potentially rendering it more privacy-intrusive at the same time; approximately 42% of apps incorporate at least two third-party trackers. On the positive side, a critical mass of apps demonstrated well-architected security practices, especially in relation to the VIN-to-owner registration process, which is a sensitive and essential matter; namely, only 18% of the apps indicated a potential issue in regard to the VIN authentication method.

Future work can concentrate on other types of either official or third-party automotive smartphone apps, say, infotainment, and possibly embrace more criteria and angles of examination, say, usability, a full-scale dynamic analysis, general data protection regulation (GDPR) level of compliance, and so forth.

Author Contributions: Conceptualization, G.K. and E.C.; methodology, G.K., V.K.; validation, E.C., G.K., and V.K.; investigation, E.C., G.K., and V.K.; writing-original draft preparation, E.C., G.K., and V.K.; writing-review and editing, E.C., G.K., and V.K.; supervision, G.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not Applicable, the study does not report any data.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

API	Application Programming Interface
APK	Android Application Package
APKiD	Android Application Identifier
App	Application
BLE	Bluetooth Low Energy

CAN	Controller Area Network
CERT	Computer Emergency Response Team
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
DoS	Denial of Service
ELF	Executable and Linkable Format binary
GDPR	General Data Protection Regulation
GOT	Global Offset Table
GRVA	Automated/Autonomous and Connected Vehicles
GUI	Graphical User Interface
IoT	Internet Of Things
ITS	Intelligent Transportation System
Jadx	Dex to Java decompiler
JSSEC	
MitM	Man-in-the-Middle
MobSF	Mobile Security Framework
NDK	Android Native Development Kit
NX	No-Execute
OAEP	Optimal Asymmetric Encryption Padding
OBD	On-Board Diagnostics
OWASP	Open Web Application Security Project
PIE	Position-Independent Executable
PNG	Portable Network Graphics
RELRO	Relocation Read-Only
SaaS	Software as a Service
SDK	Software Development Kit
T&C	Terms and Conditions
TLS	Transport Layer Security
UN	United Nations
UNECE	United Nations Economic Commission for Europe
VIN	Vehicle Identification Number
VW	Volkswagen Group
XSS	Cross-Site Scripting

References

1. Statista—The Statistics Portal for Market Data. Available online: https://www.statista.com/ (accessed on 20 February 2021).
2. Almomani, I.M.; Khayer, A.A. A Comprehensive Analysis of the Android Permissions System. *IEEE Access* **2020**, *8*, 216671–216688. [CrossRef]
3. Kato, N.; Murakami, Y.; Endo, T.; Nawa, K. Study on privacy setting acceptance of the drivers for the data utilization on the car. In Proceedings of the 14th Annual Conference on Privacy, Security and Trust — PST 2016, Auckland, New Zealand, 12–14 December 2016.
4. Damopoulos, D.; Kambourakis, G.; Anagnostopoulos, M.; Gritzalis, S.; Park, J.H. User privacy and modern mobile services: Are they on the same path? *Pers. Ubiquitous Comput.* **2013**, *17*, 1437–1448. [CrossRef]
5. Sadeghi, A.; Bagheri, H.; Garcia, J.; Malek, S. A Taxonomy and Qualitative Comparison of Program Analysis Techniques for Security Assessment of Android Software. *IEEE Trans. Software Eng.* **2017**, *43*, 492–530. [CrossRef]
6. New European Consumer Survey on Connected Car Data and Privacy. Available online: https://otonomo.io/blog/new-european-consumer-survey-on-connected-car-data-and-privacy/ (accessed on 20 February 2021).
7. Industry Report: The State of Autonomous & Connected Vehicles in 2019. Available online: https://www.automotive-iq.com/events-autonomousvehicles/downloads/the-autonomous-revolution-accelerated-industry-report-the-state-of-autonomous-connected-vehicles-in-2019 (accessed on 20 February 2021).
8. Lamssaggad, A.; Benamar, N.; Hafid, A.S.; Msahli, M. A Survey on the Current Security Landscape of Intelligent Transportation Systems. *IEEE Access* **2021**, *9*, 9180–9208. [CrossRef]
9. Mandal, A.K.; Cortesi, A.; Ferrara, P.; Panarotto, F.; Spoto, F. Vulnerability Analysis of Android Auto Infotainment Apps. In Proceedings of the 15th ACM International Conference on Computing Frontiers, Ischia, Italy, 8–10 May 2018.
10. Android Auto. Available online: https://www.android.com/auto/ (accessed on 20 February 2021).
11. Panarotto, F.; Cortesi, A.; Ferrara, P.; Mandal, A.; Spoto, F. *Static Analysis of Android Apps Interaction with Automotive CAN*; Springer International Publishing: New York, NY, USA, 2018; pp. 114–123.

12. Wen, H.; Zhao, Q.; Chen, Q.A.; Lin, Z. Automated Cross-Platform Reverse Engineering of CAN Bus Commands From Mobile Apps. In Proceedings of the 27th Annual Network and Distributed System Security Symposium—NDSS Symposium 2020, San Diego, CA, USA, 23–26 February 2020.
13. Schmittner, C.; Macher, G. Automotive Cybersecurity Standards—Relation and Overview. In Proceedings of the Computer Safety, Reliability, and Security— SAFECOMP 2019 Workshops, ASSURE, DECSoS, SASSUR, STRIVE, and WAISE, Turku, Finland, 10 September 2019.
14. UNECE. Transport Section, Working Documents (February Session). Available online: https://www.unece.org/trans/main/wp29/wp29wgs/wp29grva/grva2020.html (accessed on 20 February 2021).
15. UNECE. Cyber Security and Date Protection as well as Software Updates Proposal for a New UN Regulation on Uniform Provisions Concerning the Approval of Vehicles with Regard to Cyber Security and of Their Cybersecurity Management Systems. Available online: https://www.unece.org/fileadmin/DAM/trans/doc/2020/wp29grva/ECE-TRANS-WP29-GRVA-2020-02e.pdf (accessed on 20 February 2021).
16. UNECE. Cyber Security and Date Protection as well as Software Updates Proposal for the 01 Series of Amendments to the New UN Regulation on Uniform Provisions Concerning the Approval of Vehicles with Regard to Cyber Security and of Cybersecurity Management Systems. Available online: https://www.unece.org/fileadmin/DAM/trans/doc/2020/wp29grva/ECE-TRANS-WP29-GRVA-2020-03e.pdf (accessed on 20 February 2021).
17. UNECE. Proposal for Amendments to ECE/TRANS/WP.29/GRVA/2020/3, Draft New UN Regulation on Uniform Provisions Concerning the Approval of Vehicles with Regard to Cyber Security and of Their Cybersecurity Management Systems. Available online: https://www.unece.org/fileadmin/DAM/trans/doc/2020/wp29grva/GRVA-05-05r1e.pdf (accessed on 20 February 2021).
18. UNECE. Proposal for a New UN Regulation on Uniform Provisions Concerning the Approval of Vehicles with Regard to Software Update Processes and of Software Update Management Systems. Available online: https://www.unece.org/fileadmin/DAM/trans/doc/2020/wp29grva/ECE-TRANS-WP29-GRVA-2020-04e.pdf (accessed on 20 February 2021).
19. UNECE. Three Landmark UN Vehicle Regulations Enter into Force. Available online: https://unece.org/sustainable-development/press/three-landmark-un-vehicle-regulations-enter-force (accessed on 20 February 2021).
20. Kouliaridis, V.; Kambourakis, G.; Geneiatakis, D.; Potha, N. Two Anatomists Are Better than One-Dual-Level Android Malware Detection. *Symmetry* **2020**, *12*, 1128. [CrossRef]
21. Mobile Security Framework (MobSF). Available online: https://github.com/MobSF/Mobile-Security-Framework-MobSF (accessed on 20 February 2021).
22. Ostorlab. Mobile Application Security & Privacy Scanner. Available online: https://www.ostorlab.co/ (accessed on 8 January 2021).
23. OWASP Mobile Security Testing Guide. Available online: https://owasp.org/www-project-mobile-security-testing-guide/ (accessed on 20 February 2021).
24. Volvo On Call. Available online: https://play.google.com/store/apps/details?id=se.volvo.vcc&hl=en&gl=US (accessed on 20 February 2021).
25. BMW Connected. Available online: https://play.google.com/store/apps/details?id=de.bmw.connected.na&hl=en&gl=US (accessed on 20 February 2021).
26. My BMW. Available online: https://play.google.com/store/apps/details?id=de.bmw.connected.mobile20.row&hl=en&gl=US (accessed on 20 February 2021).
27. MINI Connected. Available online: https://play.google.com/store/apps/details?id=de.mini.connected.na&hl=en&gl=US (accessed on 20 February 2021).
28. MINI. Available online: https://play.google.com/store/apps/details?id=de.mini.connected.mobile20.row (accessed on 20 February 2021).
29. Mercedes Me. Available online: https://play.google.com/store/apps/details?id=com.daimler.ris.mercedesme.ece.android (accessed on 20 February 2021).
30. Alfa Connect. Available online: https://play.google.com/store/apps/details?id=com.fca.alfaconnect&hl=en&gl=US (accessed on 20 February 2021).
31. FIAT. Available online: https://play.google.com/store/apps/details?id=com.fca.myconnect.fiat&hl=en_US (accessed on 20 February 2021).
32. My Uconnect. Available online: https://play.google.com/store/apps/details?id=com.fca.myconnect (accessed on 20 February 2021).
33. Uconnect LIVE. Available online: https://play.google.com/store/apps/details?id=com.acn.uc (accessed on 20 February 2021).
34. MyFerrari. Available online: https://play.google.com/store/apps/details?id=ferrari.ccp.mobile (accessed on 20 February 2021).
35. FordPass. Available online: https://play.google.com/store/apps/details?id=com.ford.fordpasseu (accessed on 20 February 2021).
36. MITSUBISHI Remote Control. Available online: https://play.google.com/store/apps/details?id=com.mitsubishi_motors.remote_ps (accessed on 20 February 2021).
37. OUTLANDER PHEV Remote Ctrl. Available online: https://play.google.com/store/apps/details?id=com.inventec.iMobile2 (accessed on 20 February 2021).

38. NissanConnect. Available online: https://play.google.com/store/apps/details?id=eu.nissan.nissanconnect.services (accessed on 20 February 2021).
39. My Citroën. Available online: https://play.google.com/store/apps/details?id=com.psa.mym.mycitroen (accessed on 20 February 2021).
40. MyDS. Available online: https://play.google.com/store/apps/details?id=com.psa.mym.myds (accessed on 20 February 2021).
41. myOpel. Available online: https://play.google.com/store/apps/details?id=com.psa.mym.myopel (accessed on 20 February 2021).
42. MYPEUGEOT. Available online: https://play.google.com/store/apps/details?id=com.psa.mym.mypeugeot (accessed on 20 February 2021).
43. Jaguar InControl. Available online: https://play.google.com/store/apps/details?id=com.jaguar.incontrolremote (accessed on 20 February 2021).
44. Land Rover InControl. Available online: https://play.google.com/store/apps/details?id=com.landrover.incontrolremote (accessed on 20 February 2021).
45. Tesla. Available online: https://play.google.com/store/apps/details?id=com.teslamotors.tesla (accessed on 20 February 2021).
46. MyT. Available online: https://play.google.com/store/apps/details?id=app.mytoyota.toyota.com.mytoyota (accessed on 20 February 2021).
47. myAudi. Available online: https://play.google.com/store/apps/details?id=de.myaudi.mobile.assistant (accessed on 20 February 2021).
48. My Bentley. Available online: https://play.google.com/store/apps/details?id=uk.co.bentley.mybentley (accessed on 20 February 2021).
49. Lamborghini Unica. Available online: https://play.google.com/store/apps/details?id=lamborghini.connectedcar (accessed on 20 February 2021).
50. Porsche Connect. Available online: https://play.google.com/store/apps/details?id=com.porsche.connect (accessed on 20 February 2021).
51. SEAT CONNECT. Available online: https://play.google.com/store/apps/details?id=com.seat.connectedcar.mod2connectapp (accessed on 20 February 2021).
52. MyŠKODA. Available online: https://play.google.com/store/apps/details?id=cz.skodaauto.connect (accessed on 20 February 2021).
53. ŠKODA Connect LITE. Available online: https://play.google.com/store/apps/details?id=cz.skodaauto.connectlite (accessed on 20 February 2021).
54. We Connect. Available online: https://play.google.com/store/apps/details?id=de.volkswagen.carnet.eu.eremote (accessed on 20 February 2021).
55. Declare app permissions. Available online: https://developer.android.com/training/permissions/declaring (accessed on 20 February 2021).
56. Android API permissions. Available online: https://developer.android.com/reference/android/Manifest.permission (accessed on 20 February 2021).
57. skylot/jadx: Dex to Java decompiler—GitHub. Available online: https://github.com/skylot/jadx (accessed on 20 February 2021).
58. Mabo, T.; Swar, B.; Aghili, S. A Vulnerability Study of Mhealth Chronic Disease Management (CDM) Applications (apps). In Proceedings of the WorldCIST'18—6th World Conference on Information Systems and Technologies, Naples, Italy, 27–29 March 2018.
59. Imai, H.; Kanaoka, A. Chronological Analysis of Source Code Reuse Impact on Android Application Security. *J. Inf. Process.* **2019**, *27*, 603–612. [CrossRef]
60. Research Results on SHA-1 Collisions. Available online: https://csrc.nist.gov/News/2017/Research-Results-on-SHA-1-Collisions (accessed on 20 February 2021).
61. Janus CVE. Available online: https://nvd.nist.gov/vuln/detail/CVE-2017-13156 (accessed on 20 February 2021).
62. Network security configuration. Available online: https://developer.android.com/training/articles/security-config (accessed on 20 February 2021).
63. Android Application Identifier for Packers, Protectors, Obfuscators and Oddities—PEiD for Android. Available online: https://github.com/rednaga/APKiD (accessed on 20 February 2021).
64. Duan, Y.; Zhang, M.; Bhaskar, A.V.; Yin, H.; Pan, X.; Li, T.; Wang, X.; Wang, X. Things You May Not Know About Android (Un)Packers: A Systematic Study based on Whole-System Emulation. In Proceedings of the 25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, CA, USA, 18–21 February 2018.
65. 2020 CWE Top 25 Most Dangerous Software Weaknesses—CWE Mitre. Available online: https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html (accessed on 20 February 2021).
66. Google Official Developers Android Webpage - Security Tips. Available online: https://developer.android.com/training/articles/security-tips (accessed on 20 February 2021).
67. Exodus Privacy. Available online: https://exodus-privacy.eu.org/en/ (accessed on 20 February 2021).

68. Razaghpanah, A.; Nithyanand, R.; Vallina-Rodriguez, N.; Sundaresan, S.; Allman, M.; Kreibich, C.; Gill, P. Apps, Trackers, Privacy, and Regulators: A Global Study of the Mobile Tracking Ecosystem. In Proceedings of the 25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, CA, USA, 18–21 February 2018.
69. Vallina-Rodriguez, N.; Sundaresan, S.; Razaghpanah, A.; Nithyanand, R.; Allman, M.; Kreibich, C.; Gill, P. Tracking the Trackers: Towards Understanding the Mobile Advertising and Tracking Ecosystem. *arXiv* **2016**, arXiv:1609.07190.
70. Liu, X.; Liu, J.; Zhu, S.; Wang, W.; Zhang, X. Privacy Risk Analysis and Mitigation of Analytics Libraries in the Android Ecosystem. *IEEE Trans. Mob. Comput.* **2020**, *19*, 1184–1199. [CrossRef]
71. AltBeacon/Android-Beacon-Library. Available online: https://github.com/AltBeacon/android-beacon-library (accessed on 20 February 2021).
72. Geofencing-Geo-Targeting-Beaconing. Available online: https://www.braze.com/blog/geofencing-geo-targeting-beaconing-when-to-use/ (accessed on 20 February 2021).
73. AppDynamics: An APM solution. Available online: https://www.appdynamics.com/ (accessed on 20 February 2021).
74. Branch—A mobile linking platform. Available online: https://branch.io/ (accessed on 20 February 2021).
75. Leanplum. Available online: https://www.leanplum.com/ (accessed on 20 February 2021).
76. HockeyApp. Available online: https://appcenter.ms/ (accessed on 20 February 2021).
77. Demdex. Available online: https://www.adobe.com/analytics/audience-manager.html (accessed on 20 February 2021).
78. Adobe Buys Behavioral Data Management Platform DemDex. Available online: https://techcrunch.com/2011/01/18/adobe-buys-behavioral-data-management-platform-demdex/ (accessed on 20 February 2021).
79. Appcenter Android Crashes. Available online: https://docs.microsoft.com/en-us/appcenter/sdk/crashes/android (accessed on 20 February 2021).
80. Jumio: End-to-End ID and Identity Verification Solution. Available online: https://www.jumio.com/ (accessed on 20 February 2021).
81. Airship | Customer Engagement Platform. Available online: https://www.airship.com/ (accessed on 20 February 2021).
82. Gigya | Customer Data Management. Available online: https://www.sap.com/products/crm/customer-data-management.htm (accessed on 20 February 2021).
83. Android SDK—Facebook for Developers. Available online: https://developers.facebook.com/docs/android (accessed on 20 February 2021).
84. MoPub: Powerful App Monetization. Available online: https://www.mopub.com/en (accessed on 20 February 2021).
85. Salesforce: We Bring Companies and Customers Together. Available online: https://www.salesforce.com/products/marketing-cloud/overview/ (accessed on 20 February 2021).
86. Adobe Experience Cloud. Available online: https://www.adobe.com/experience-cloud.html (accessed on 20 February 2021).
87. Countly. Available online: https://count.ly/ (accessed on 20 February 2021).
88. Countly Server Github. Available online: https://github.com/Countly/countly-server (accessed on 20 February 2021).
89. Splunk MINT. Available online: https://mint.splunk.com/ (accessed on 20 February 2021).
90. Dynatrace. Available online: https://www.dynatrace.com/ (accessed on 20 February 2021).
91. Batch. Available online: https://batch.com/ (accessed on 20 February 2021).
92. Adjust. Available online: https://www.adjust.com/ (accessed on 20 February 2021).
93. Optimizely: The World's Leading Digital Experience Platform. Available online: https://www.optimizely.com/ (accessed on 20 February 2021).
94. App Manifest Overview | Android Developers. Available online: https://developer.android.com/guide/topics/manifest/manifest-intro (accessed on 20 February 2021).
95. Compromising Android Applications with Intent Manipulation. Available online: https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/compromising-android-applications-with-intent-manipulation/ (accessed on 20 February 2021).
96. Understand Tasks and Back Stack | Android Developers. Available online: https://developer.android.com/guide/topics/manifest/activity-element#aff (accessed on 20 February 2021).
97. Content Provider Basics | Android Developers. Available online: https://developer.android.com/guide/topics/providers/content-provider-basics (accessed on 20 February 2021).
98. Launch Mode | Android Developers. Available online: https://developer.android.com/guide/topics/manifest/activity-element (accessed on 20 February 2021).
99. Hwang, S.; Lee, S.; Ryu, S. All about activity injection: Threats, semantics, detection, and defense. *Softw. Pract. Exp.* **2020**, *50*, 1061–1086. [CrossRef]
100. Google Official Developers Android Webpage—Android NDK. Available online: https://developer.android.com/ndk (accessed on 20 February 2021).
101. Li, L.; Bissyandé, T.F.; Klein, J.; Traon, Y.L. An Investigation into the Use of Common Libraries in Android Apps. In Proceedings of the IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, 14–18 March 2016; Volume 1, pp. 403–414. [CrossRef]
102. Taylor, V.F.; Beresford, A.R.; Martinovic, I. Intra-Library Collusion: A Potential Privacy Nightmare on Smartphones. *arXiv* **2017**, arXiv:1708.03520.

103. Backes, M.; Bugiel, S.; Derr, E. Reliable Third-Party Library Detection in Android and its Security Applications. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016.
104. RELRO: RELocation Read-Only. Available online: https://medium.com/@HockeyInJune/relro-relocation-read-only-c8d0933faef3 (accessed on 20 February 2021).
105. Derr, E.; Bugiel, S.; Fahl, S.; Acar, Y.; Backes, M. Keep me Updated: An Empirical Study of Third-Party Library Updatability on Android. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, 30 October–3 November 2017.
106. Salza, P.; Palomba, F.; Nucci, D.D.; D'Uva, C.; Lucia, A.D.; Ferrucci, F. Do developers update third-party libraries in mobile apps? In Proceedings of the 26th Conference on Program Comprehension, ICPC 2018, Gothenburg, Sweden, 27–28 May 2018.
107. NVD NIST. Available online: https://nvd.nist.gov/ (accessed on 20 February 2021).
108. SQLite. Available online: https://www.sqlite.org/index.html (accessed on 20 February 2021).
109. OpenSSL. Available online: https://www.openssl.org/ (accessed on 20 February 2021).
110. libpng. Available online: http://www.libpng.org/pub/png/libpng.html (accessed on 20 February 2021).
111. openCV. Available online: https://opencv.org/ (accessed on 20 February 2021).
112. jQuery. Available online: https://jquery.com/ (accessed on 20 February 2021).
113. InAppBrowser. Available online: https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-inappbrowser/index.html (accessed on 20 February 2021).
114. zlib. Available online: https://zlib.net/ (accessed on 20 February 2021).
115. libexpat. Available online: https://libexpat.github.io/ (accessed on 20 February 2021).
116. libcurl. Available online: https://curl.se/libcurl/ (accessed on 20 February 2021).
117. libjpeg. Available online: http://libjpeg.sourceforge.net/ (accessed on 20 February 2021).
118. VIN Decoder - Decoding VIN numbers. Available online: https://www.vindecoder.pl/ (accessed on 20 February 2021).
119. What Is Dataplug? Available online: https://www.john-clark.co.uk/volkswagen/latest-news/vw-data-plug/ (accessed on 20 February 2021).
120. NIST Password Policy. Available online: https://pages.nist.gov/800-63-3/sp800-63b.html (accessed on 20 February 2021).
121. Kambourakis, G.; Draper-Gil, G.; Sanchez, I. What Email Servers Can Tell to Johnny: An Empirical Study of Provider-to-Provider Email Security. *IEEE Access* **2020**, *8*, 130066–130081. [CrossRef]
122. Temporary Mail. Available online: https://temp-mail.org/en/ (accessed on 20 February 2021).