Introduction to being a Privacy Detective: Investigating and Comparing Potential Privacy Violations in Mobile Apps Using Forensic Methods

Stefan Kiltz and Robert Altschaffel and Thorsten Lucke and Jana Dittmann

Otto von Guericke University Magdeburg
Magdeburg, Germany
Email: Kiltz@iti.cs.uni-magdeburg.de

***Abstract*—This paper discusses means to evaluate the potential impact of data flows caused by the use of smartphone apps (*app*lications) on the privacy of the user. While the data flows are often caused by trackers, permissions set the framework on which data can flow between the smartphones and the remote party. Hence, we devise a concept to examine privacy violations caused by trackers and permissions in mobile apps and to render the results of said examination more comparable and reliable based on the characteristics of the examination methods (custody, examined forensic data streams and type of communication). We define two different examination scenarios in which this approach can be deployed and conduct practical tests in these two scenarios. For the first scenario, the concept is applied to the static evaluation of 8 exemplary mobile apps running on the Android platform using 3 different methods (*Exodus Privacy*, *Exodus Standalone* and *AppChecker*) identifying 162 permissions and 42 trackers in total. The second scenario employs these three methods in order to examine the extent to which three mobile browsers reveal information towards the respective developers. Our main contributions are the application of a model of the forensic process to the examination of the loss of potential privacy due to the use of mobile apps in order to provide comparability of the findings. In addition, a proposal for a visualization scheme capable of displaying test results from privacy examinations covering a large number of examination items is proposed.**

Keywords–Privacy Measurement; Data sovereignty.

I. INTRODUCTION

Privacy and data protection are very relevant topics from a legal and data sovereignty perspective. While the legal perspective is no part of this paper, the *Principles relating to processing of personal data* of Article 5 of the GDPR[1] provide a useful overview on the topic of privacy related data. These principles include *data minimisation*, *storage limitation* and *lawfulness, fairness and transparency* - principles often not used in app deployment, most apps contain trackers [2].

This paper aims at supporting privacy and data protection by providing the user with methods to identify data flows caused by the use of mobile apps (executed on the smartphone), which could threaten the privacy. Knowledge about these data flows is essential to obtain some degree of data sovereignty. These data flows could be used by third parties to identify customers, create profile, send targeted advertisement (including those leading to generally known negative social consequences like advertisement for alcohol or micro-targeting during political campaigns) or exclude customers from services based on their liquidity. In addition, these trackers use unnecessary resources (e.g., CPU power, bandwidth, energy) without a benefit to the user and the environment. This increased need for resources also leads to a more negative global ecological footprint.

The discussion whether certain data flows violate the right of privacy of an user relies on legal background and a review of the relevant laws. However, this paper focuses on a purely technical approach based on the technical properties we can identify. However, knowledge about the presence of these data flows is by itself a necessary requirement to achieve data sovereignty. This is of relevance when deciding whether an app is appropriate for a given use case, e.g., the use in an educational or corporate setting. In these scenarios, a teacher or IT specialist could be the data detective identifying various trackers.

This knowledge can be used to prevent privacy-related leakage. Users can prevent trackers by using various blocking mechanisms like NoScript [3] or the use of a Pi-hole [4]. However, these mechanisms all have their limitations and require installation and configuration for use. The knowledge about tracker detection techniques has other uses; developers of apps can also use the knowledge about trackers within their apps to remove such trackers, introduced by e.g., software development kits or libraries.

A necessary property to identify trackers is the destination of a given data flow. During the course of this paper, we identify two different potential destinations for data flows common in the use of mobile apps:

- Data flow to the service provider (DF_{fp})
- Data flow to a third party (DF_{tp})

The second property catches the aspect of whether the data flow is necessary in order for the app to provide the functionality. Either the app is able to provide the functionality without this data data or it is not. Hence, we can further specify the potential data flows:

- Data flow to the service provider necessary for the functionality of the app ($DF_{fp.req}$)
- Data flow to the service provider not necessary for the functionality of the app ($DF_{fp.nrq}$)
- Data flow to a third party necessary for the functionality of the app ($DF_{tp.req}$)
- Data flow to a third party not necessary for the functionality of the app ($DF_{tp.nrq}$)

A common example for those data flows unnecessary to provide the functionality intended by the user include *trackers*.

 ISBN: 978-1-61208-821-1

These trackers are embedded into many apps in order to gather data about the user and the use patterns of the given tools. This is often used as a foundation for the identification of users which can in turn be used for ads targeted at specific users. These trackers pose a risk to the privacy of the user. In *disconnect-tracking-protection* [5] a detailed description based on a technical review is given as to why a specific item is listed as a tracker and might be used as guideline to discern whether the data flows identified using the approach in this paper are threats to privacy.

For the sake of simplicity, we refer to any data flow not necessary to provide the functionality intended by the user as a *tracker* during the course of this paper. Various tools designed to identify these trackers are available with an overview on these provided in Section II-C. A fundamental problem is that the results provided by these tools are not comparable. This increases the difficulty of verifying the results provided by one such tool through the use of a different tool. This is due to different underlying criteria used for the identification of trackers and a varying degree of documentation provided to the examiner by these tools. This paper explores the use of a structured investigative approach as used in the field of computer forensics to achieve this comparability in finding unnecessary data flows.

However, the absence of any unnecessary data flows does not guarantee the absence of risk for the user's privacy. For example, the service provider could aggregate $\mathrm{DF}_{fp.req}$ or $\mathrm{DF}_{tp.req}$ over the course of various requests and compile them to form an user profile. However, the data that could be transferred is limited by the access the specific app has to the operating system. This is controlled by permissions. Hence, identifying these permissions offers additional information relevant to judge potential data protection violations.

In general, we identify two different scenarios in which the approach presented in this paper can be used:

- Examination Scenario 1 (**ES1**): one (or multiple) app(s) are examined for trackers and permissions
- Examination Scenario 2 (**ES2**): various alternative apps for a specific use case are compared with regards to potential privacy violations

Due to the complexity of this topic this paper focuses on the identification of the four formerly defined data flows during the use of mobile apps as well as the permissions used by these apps. In order to achieve this goal, this paper is structured as follows: Section II gives an overview on the domain investigated in this paper, the current tools used to identify trackers and permissions in mobile apps and the structured approach to perform an investigation as used in the field of computer forensics. Section III applies the structured approach of a forensic investigation to the identification of trackers (and data flows in general). This includes a discussion on how the tools detecting trackers and identifying permissions work in detail and how this type of examination is currently conducted showing the challenges of the specific domains in question. Section IV describes the creation of a testbed which includes the tool sets necessary to conduct such an investigation following the structured approach. It also shows how this testbed is then used to identify and categorize various data flows during the use of eight selected exemplary mobile apps in **ES1**. In addition, three exemplary selected different browser are compared as an example for **ES2** providing a more in-depth comparison on what the achieved results reveal about potential violation of the user privacy by the use of these apps. This paper closes with an overview and an outlook provided in Section V.

II. FUNDAMENTALS

This section provides some background helpful to follow the discussions laid out during the subsequent chapters. An overview on the various approaches to evaluate the privacy of websites and apps is provided. This includes the identification of various data flows. In addition, the properties specific to apps in the mobile domain affecting the identification of trackers are explored.

A. Exemplary selection of approaches to evaluate the privacy of websites and apps

Several approaches to determine the privacy and IT security of websites exist. In [6] a system to improve the privacy and IT security is described, which analyses selected privacy and IT security aspects. A special feature of the resulting analysis site *privacyscore.org* [7] is that arbitrary users can supply a single URL to be tested or a list thereof (crowd-sourced list). The system design allows revisiting those existing URL(s) and thus enables an analysis of the privacy and IT situation of said URL(s) over time and for privacyscore.org users alike.

One empirical study to systematically determine the privacy and IT security of apps for the android operating environment is described in [2]. It reveals that most apps contain trackers, that the tracking is category-sensitive and a highly trans-national phenomenon. A guideline on how Apps are tested for trackers is provided by various sources. One example includes *mobilsicher.de* [8] which provides an outline of their approach in German.

B. Mobile Apps

The topics discussed in this paper are also relevant in the domain of desktop computer systems. Indeed, some of the concepts presented in this paper can be transferred to this domain. However, for the sake of brevity, this paper focuses on the mobile domain in order to discuss some of the specific challenges of this domain when investigating potential privacy leaks. Hence, for the course of this paper, an app(*lication*) refers to any type of application that is executable on a smartphone and which is distinguishable from the operating environment (for example in that it is downloaded or updated separately).

Apps use the resources of the smartphone as provided by an underlying operating system. During the course of this paper, all apps examined and methods used are running on the Android platform [9]. This does not represent any inclination of the authors towards this platform nor should it indicate that the methods used in this paper are not applicable to different platforms. In the case of Android operating systems, the access of the apps is restricted by the use of various permissions. Permissions describe the access rights granted for a specific app. In essence, they restrict access to given information. Examples include the access to camera or the address book. Hence, these permissions are useful to discern which information could be communication within the specific data flows.

 ISBN: 978-1-61208-821-1

C. Methods to identify data flows

Identifying data flows forms the foundation of identifying potential privacy violations. Two principal approaches exist to identify such data flows in mobile apps. These approaches are implemented in various tools but these tools represent insulated solutions rather than a comprehensive approach providing comparable results.

a) Static Analysis: Static Analysis describes analysis performed by investigating the binary representation of an app. This binary representation can be interpreted in order to identify certain functions. Trackers for example usually employ a set of specific function calls or employ communication to certain known servers. Such patterns are referred to as signatures. As long as the signature for a given tracker is known, it can be identified. The complexity of this interpretation depends on the complexity of the programming technology used to develop the app. Some apps might use standardized development tools while others might have their binary formats obfuscated to prevent such analysis.

A tool implementing static analysis of mobile apps is *Exodus Privacy* [10]. The characteristics and the operations methods of this tool are described in III-B. The tool is then used during the case study as described in IV-B providing insight into its use and usefulness.

b) Dynamic Analysis: Dynamic Analysis investigates the runtime behaviour of an app during its execution. In the context of this paper, this mostly includes the communication behaviour of the app. Hence, the data flows are observed while they occur. A tool useful for the capturing of data flows is *Wireshark* [11].

c) Specifics of mobile apps relevant during examination: There are some factors which impact the capabilities to examine apps in the mobile domain:

- **Prop1**: *large amount of background processes*: in general, many background processes are active on a mobile system implying the presence of background noise which must be taken into account
- **Prop2**: *very low control over operating system*: mobile operating systems generally limit the user in the access to various system information
- **Prop3**: *standardization of development tools*: apps are developed using standard frameworks and languages which makes analysis easier
- **Prop4**: *reliance on system functions*: apps often use libraries and operating system calls which are protected or obfuscated against analysis
- **Prop5**: *apps contain a manifest*: this gives some information about the use of certain system permission by the app, but practical tests have shown that these are not necessarily conclusive
- **Prop6**: *various variants*: there are usually various variants of apps in the mobile domain compiled for various architectures and operating systems with a potentially different behaviour in regards to data flows
- **Prop7**: *App bundles*: sometimes these different variants are combined into an App Bundle which includes various resources necessary in order to create a device specific installation

Prop1 and **Prop2** have a negative impact on the capabilities to perform dynamic analysis in the context of mobile apps, while **Prop5** eases the complexity of of detecting permissions during static analysis. **Prop6** and **Prop7** raise the difficulty of obtaining the correct binary for analysis in the first place. Here, the use of checksums and other methods in order to identify the specific binaries is necessary.

D. The structured approach of Computer Forensics

According to [12] computer forensics is *The use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations*.

Such an approach implies the use of proven methods in a structured process in order to document various traces with the overall aim to present them in a manner which enables a person reviewing the result to form a well-founded conclusion based on these traces. When investigating potential privacy violations during the use of mobile apps the same traceability and comparability of the results is intended. Hence, methods from the field of computer forensics could potentially be used to achieve this.

In general, computer forensic investigations follow a structured process. This process is often described by forensic process models. During the course of this paper, we employ a forensic process model based on the one described in [13] (which is based on [14]) as it offers some advantages for the task at hand.

This forensic process model structures the forensic process along the lines of six *Investigation Steps*. A main advantage of this forensic process model is the inclusion of a Strategic Preparation (**SP**) which covers *measures taken by the operator of an IT-system in order to support a forensic investigation prior to an incident* [14].

The forensic traces originate from three distinct forensic *Data Streams* as described in [15]. These data streams identify the various sources of forensic evidence within the forensic process and assign general properties to these sources:

- Non-volatile Memory: *Memory inside a computing unit which maintains its content after the unit is disconnected from its respective power supply.* (denoted as DS_T in this paper)
- Volatile Memory: *Memory inside a computing unit which loses its content after the unit is disconnected from its respective power supply.* (denoted as DS_M)
- Communication: *All the data transmitted to other computing units via communication interfaces.* (denoted as DS_N)

The potential traces relevant during the forensic investigation are categorized into nine *Data Types* based on how the respective data is handled during the forensic process. The following relevant Data Types are identified in [13]:

- hardware data (**DT1**): *Data in a computing unit which is not, or only in a limited way, influenced by software.*

 ISBN: 978-1-61208-821-1

- raw data (**DT2**): *A sequence of bits within the data streams of a computing systems not (yet) interpreted.*
- details about data (**DT3**): *Data added to other data, stored within the annotated chunk of data or externally*
- configuration data (**DT4**): *Data which can be changed by software and which modifies the behaviour of software and hardware, excluding the communication behaviour*
- network configuration data (**DT5**): *Data that modifies system behaviour with regards to communication*
- process data (**DT6**): *data about a running software process within a computing unit*
- session data (**DT7**): *data collected by a system during a session, which consist of a number of processes with the same scope and time frame*
- application data (**DT8**): *data representing functions needed to create, edit, consume or process content relied to the key functionality of the system*
- functional data (**DT9**): *data content created, edited, consumed or processed as the key functionality of the system*

Six *Classes of Methods* describe the requirements for the use of the various forensic methods by categorizing them based on what kind of software provides said method. This corresponds to the use of specific tools (or tool chains) in order to investigate a certain type of data. Some of these classes of methods defined in [14] are relevant for this paper. These are the Operating system (**OS** - *methods provided by the operating system [...]*), Explicit means of intrusion detection (**EMID** - *methods provided by additional software [...] being executed autonomously on a routine basis and without a suspicion of an incident*), IT application (**ITA** - *methods provided by IT-Applications that are operated by the user [...]*) and Scaling of methods for evidence gathering (**SMG** - *methods to further collect evidence to be used if a suspicion is raised [...]*).

III. Structured approach to investigate and compare potential privacy violations in mobile apps

In this section, the approach for digital forensics discussed in Section II-D is applied to achieve comparability between the various methods to examine third and first party tracking employed in mobile apps.

A. System landscape analysis and its resulting forensic process as part of Strategic Preparation (*SP*) and its impact on comparability

The investigation step of Strategic Preparation (see Section II-D) plays a decisive role during this process in trying and testing the measures ahead of an incident and setting the system boundaries of systems which are examined and those used for examination. Hence, this step directly impacts the results and their credibility. Setting these boundaries is done during a *system landscape analysis*. Such an analysis for our use case of identifying third and first party tracking by apps via employing static analysis is depicted in Figure 1. It also introduces a possible extension enabling the analysis of websites. The system landscape analysis allows for the identification of various characteristics in which the employed

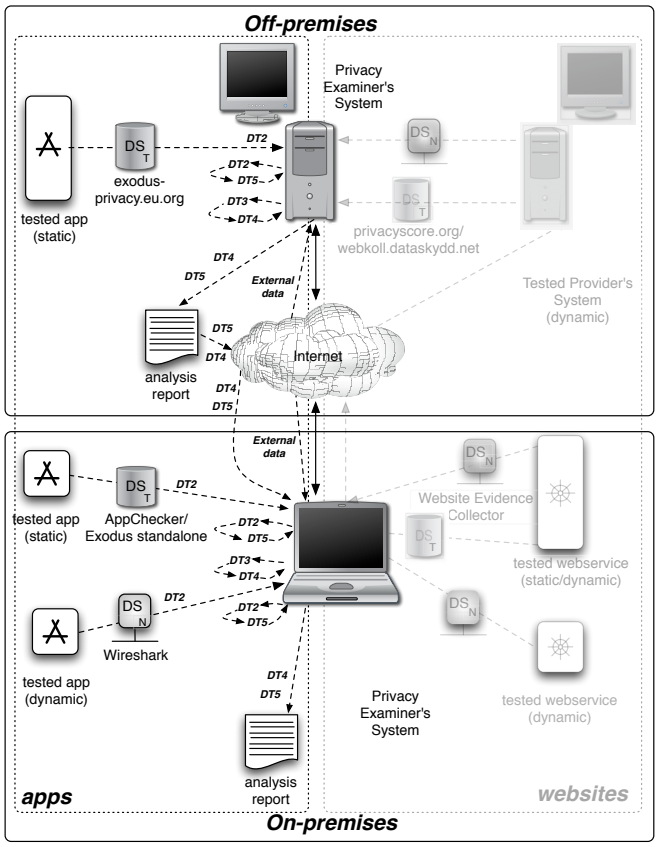

Figure 1. System landscape analysis of the examination of apps and websites regarding first and third party tracking using the categorization into *custody*, *data streams*, and *type of examination*

methods differ. These characteristics carry some implications on how these methods can be used in order to identify third and first party tracking. This affects the credibility of the results.

The first identified characteristic is *custody*. This characteristic encompasses two aspects. These aspects are *custody over the method* used for the examination and *custody over the examination item* which is an app in the context of this paper. This custody can either be *on-premises* (the examiner has custody over this component) or *off-premises* (another entity has custody over this component). If two different methods share the same characteristic in terms of custody, we refer to them as *intra-premises*. If they differ in this respect, we refer to them as *inter-premises*. Generally speaking, having custody of these two aspects implies a greater control over them and, hence, a higher credibility. Secondly, we identify the *data stream* examined by the given method as another relevant characteristic (see II-D for an introduction into the various forensic data streams). A method could analyse the binary obtained from the non-volatile memory (DS_T), the volatile memory obtained during the execution of a binary (DS_M) or the communication observed during execution (DS_N). Finally, we can categorize the *type of examination* into a *static examination* (a single snapshot of the behaviour of the website or the app) and a *dynamic examination* (a change in behaviour of an app over time).

With our stated goal of achieving comparability of tools and services for privacy examination, i.e., the sets of methods

from the model from [14], this categorization provides a structure supporting the comparability between various methods. This is not affected whether these methods rely on the same internal engine (*intra-engine*) or not (*inter-engine*).

B. Comparison of methods within the forensic framework

The forensic data types shown in Section II-D are generally applicable to all data streams (see Section III-A and can be used to indicate privacy issues (primarily by their presence but also by their absence). Also they can be used to describe the methods employed to identify and examine data flows and their respective actions in a comparable manner.

One such example is shown in Table I. This table provides an overview on the three methods used for identifying trackers and permission during this paper. It includes an intra-engine comparison of two different means to employ the *Exodus Privacy Engine* and well as an inter-engine comparison to a third method.

The two means to employ the *Exodus Privacy Engine* are *Exodus Privacy* ([10], in short: *EPO*) and *Exodus Standalone* ([16], in short: *ESA*). *EPO* is a web-based service which runs remotely (*off-premises*). The user sends a request to the service which then downloads an app (in form of an .APK file) from the Google Play Store. Neither the method itself nor the examination item are in the *custody* of the examiner. *ESA* is a console application which runs on a local system and is supplied with a local .APK file to examine (*on-premises*). In this case, method and examination item in the *custody* of the examiner. Both variants examine a binary file which has to be downloaded and stored before the examination, hence the *data stream* DS_T is examined. Both employ static measures as its *type of examination*.

AppChecker ([17], in short: *APC*) provides an inter-engine comparison. This tool is a console application running on a local system (*on-premises*) with the method and examination item being in the *custody* of the examiner. Again, it employs static measures (*type of examination*) on a binary file (DS_T).

With this in mind, a closer look on the exact nature of the examination performed by these three methods is necessary with a specific focus on comparability and documentation. In both cases, access to the specific app (in form of an .APK) is required. In the case of *ESA* and *APC*, this .APK might already be available so the first action is optional. Whether the .APK has to be downloaded first (in case of *EPO*), copied from another medium or is already present on the examiners system, it represents raw data (**DT2**). This .APK is then extracted to reveal the binary (**DT2**) and the manifest which represents information about this binary (**DT3**). Following this, various pieces of information are extracted from the present data. The binary (**DT2**) is searched for hosts and ip addresses leading to a list of potential communication partners (representing **DT5**). The manifest (**DT3**) is investigated in order to compile a list of the various permissions the app is supposed to be granted by the system it runs on (representing **DT4**). In the next step, both lists are compared to *external data* which does not originate from the examined source but contributes to the examination result. This external data represents signatures which could include hosts known to be used for advertisement. In essence, these data represents a list of known trackers (in case of host names or ip addresses) or unwanted permissions. This leads to a reduced list of relevant trackers (**DT5**) or relevant permissions (**DT4**). In the final step, this data is then compiled into a report.

While the operations are identical for each three methods, the visibility to the examiner is different. In the case of *ESA* and *APC* every action was performed locally and was hence observable to the investigator. At every action, inputs and outputs could be documented in detail. In the case of *EPO* every action taking place after the process is started is not observable for the examiner. The examiner has to rely on the summary provided by *EPO* after the method is finished. However, the use of *EPO* requires far less effort from the examiner in terms of required resources.

TABLE I. FORENSIC *DATA TYPES* PROCESSED DURING THE INTERNAL ACTIONS AND THEIR VISIBILITY TO THE EXAMINER DURING STATIC ANALYSIS

Internal Action	*Data Types*	observable in ...		
		EPO - [10]	ESA - [16]	APC - [17]
Download .APK	**DT2**	✗	✓	✓
Extract .APK	**DT2**, **DT3**	✗	✓	✓
Binary: Extract Hosts	**DT2** → **DT5**	✗	✓	✓
Manifest: Extract Permission	**DT3** → **DT4**	✗	✓	✓
Host: Compare	**DT5**, *ext* → **DT5**	✗	✓	✓
Manifest: Compare	**DT4**, *ext* → **DT4**	✗	✓	✓
Generate Report	**DT4**, **DT5** → *Report*	✓	✓	✓

C. Visualization of examinations results

Since the tools and services (i.e., sets methods from [14]) typically gather a large number of results, an efficient visualization of the results of their usage is paramount. This visualization should support an easy comparison of different methods and should be easily extensible with regards to new *result categories* (horizontal view in Figure 2) and new tested apps and websites (vertical view).

We choose a DNA-graph-style representation as it allows for the additional integration of information regarding the data type (e.g., URLs of known trackers as **DT3** in the the network data stream DS_N, app permissions as configuration data **DT4**) using colouring/shading. A very important property of this visualization type is that it also depicts the *absence* using marked positions, which greatly supports the comparison of different methods to examine potentially privacy violating data flows.

IV. CASE STUDY

This section describes the creation of a test environment in order to apply various methods to examine potentially privacy violating data flows in a sample of selected apps (**ES1**) or to examine various alternative apps usable for web browsing with regards to potential privacy violations (**ES2**).

A. Building a test environment

A test environment should fulfill a number of requirements that apply to both dynamic and static tests (see Section II). To ensure the authenticity of the tested app, it should be acquired from the official distribution system. One means to access these stores and extract the app as it was downloaded is to use an Android Emulator such as the Emulator from the official *AndroidSDK* [18] or using dedicated emulators such as *Genymotion* [19]. If the option of shared access to

 ISBN: 978-1-61208-821-1

the emulated system is used, access to the official .APK of a given app is possible. Thus, the app can be downloaded from the official distribution and yet is accessible to the tools for static evaluation by means of accessing the mass storage of the emulated system. *AppChecker* and *Exodus Standalone* as well as *Exodus Privacy* compute a SHA265 cryptographic hash, enabling also an integrity check. The maintenance of both security aspects are vital for the forensic process (see Section II-D).

Although dynamic analysis is only included in a minor role in this paper for the sake of brevity, a few short notes for a successful conduct of such a structured examination shall be provided. In this case, the creation of a "low-noise" environment is of great help for the examiner. Since in dynamic analysis the network traffic is analysed, any interference from other parts of the Android operating system should be minimized. While we consider a complete exclusion of all surplus network traffic impossible with current mobile operating systems, at least a reduced interference should be achieved by the removal of all apps and services not needed to execute a given app for testing.

*B. (**ES1**): Testing eight different apps for potential privacy violations*

For the tests a set of eight different apps suited for use in an educational institution was selected. The tests were conducted using the guidelines provided in Section IV-A. *Genymotion 3.1.1* [19] was used on a Lenovo T61 Laptop with 4GB RAM running *Ubuntu18.04* [20]. The Android emulator was run within a virtualized system (realized using *VirtualBox* [21] with the necessary configuration for shared access to the virtual file system. The respective apps where then downloaded from the respective Playstore on the emulated phone (emulating an Android 7.1). The downloaded binary file was then transferred using the *Terminal App* [22] to the shared folder. Here, the sha256sum for the specific binary was calculated and documented.

This binary was then examined using *Exodus Standalone* and *AppChecker*. The execution of both methods can be documented on the command line and the outputs created can be stored. Since the SHA256 checksum of the binaries are known, the results of these two methods running *on-premises* can be compared to the results provided by the *off-premises* method of *Exodus Privacy*. The three used methods are explained in detail in Section III-B including the use of forensic data types during the application of these methods.

The execution of all methods on the eight selected specimen was successful. Based on the considerations for the visualization made in Section III-C, the results are shown in Figure 2. It shows the three different methods of investigations for each of the eight different specimen as the rows while the columns show the different identified trackers (**DT5**) and the identified permissions (**DT4**). A colored box indicates that the specific tracker or permission was identified within the specific specimen using the specific method.

For the trackers, it is notable that there are very few *intra-engine* differences between *EPO* and *ESA*. The exception is *Moodle*, where *EPO* identifies the Google Firebase Analytics tracker and *ESA* does not. This might be due to a difference in the list of known tracker signatures between these versions. However, since no information on the exact version of this list used is provided by the two methods, a further examination is not possible. While *ESA* could be modified to store this list of signatures for documentation purposes (due to being open source and on-premises), this is not possible for *EPO*. Generally, both methods provide the same results, albeit the use of *ESA* provides for a better documentation on how these results are achieved. There are, however, some notable *inter-engine* differences in the results when compared to *APC*. This can be seen, for example, with DropBox, or Shazam. The reason here are different heuristics (including the signatures for the identification of trackers). An overview on these results is provided in Table II. In general, there is usually a core of trackers for any given app identified by all methods. Some trackers are only found by *EPO* and *ESA* and some only by *APC*. A review of the identified trackers show that all of them fall under DF_{tp}) since they represent connections to a third party. Based on external knowledge about the specific names of the identified trackers, these most likely all represent $DF_{tp.nrq}$.

TABLE II. NUMBER OF IDENTIFIED TRACKERS DURING **ES1** ON EIGHT SELECTED APPS AS PERFORMED IN SECTION IV-B

Application Name and Version	*ESA*	*EPO*	*APC*	common to all
Corona-Warn 1.2.1	0	0	1	0
Dropbox 194.2.6	5	5	4	2
GuitarTuna 6.4.0	10	10	11	9
Moodle 0.0.0	0	1	2	0
Pixabay 1.1.3.1	2	2	2	0
QR & Barcode Scanner 2.1.32	5	5	6	4
Shazam 10.38.0-200709	4	4	5	2
Signal 4.69.4	0	0	1	0

TABLE III. NUMBER OF IDENTIFIED PERMISSIONS DURING **ES1** ON EIGHT SELECTED APPS AS PERFORMED IN SECTION IV-B

Application Name and Version	*ESA*	*EPO*	*APC*	common to all
Corona-Warn 1.2.1	8	8	8	8
Dropbox 194.2.6	23	23	23	23
GuitarTuna 6.4.0	9	9	9	9
Moodle 0.0.0	30	30	30	30
Pixabay 1.1.3.1	9	9	9	9
QR & Barcode Scanner 2.1.32	13	13	13	13
Shazam 10.38.0-200709	14	14	14	14
Signal 4.69.4	65	65	65	65

For the permissions, the results are identical for all methods applied to all apps as can be seen in Table II. This is based on the reason that same approach for detecting permissions is used by all these methods. All the methods discussed here perform a review of the manifest to identify trackers.

*C. (**ES2**): Examining various alternative apps usable for web browsing with regards to potential privacy violations*

For this test, three exemplary selected apps for web browsing are examined with regards to potential privacy violations. The three mobile browsers are *Chrome*, *F-Droid Fennec* and *Mozilla Firefox*.

The tests employed the same examination setup as used in **ES1** (see Section IV-B). This includes the three methods *Exodus Standalone*, *AppChecker* and *Exodus Privacy* used for

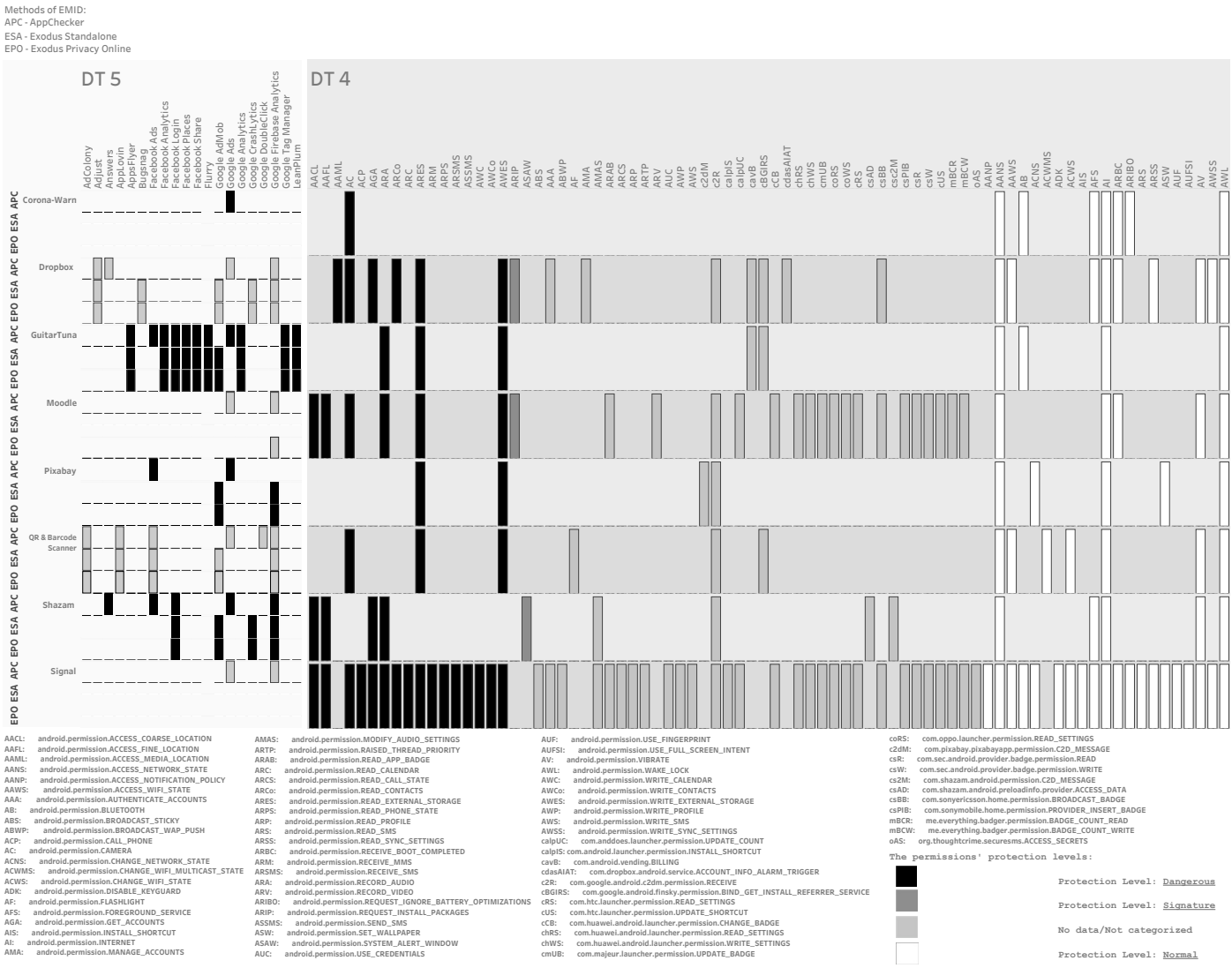

Figure 2. Trackers and permissions identified during **ES1** on eight selected apps as performed in Section IV-B

static analysis. These three methods and the forensic data types used during their application are discussed in detail in Section III-B. However, in this case dynamic analysis was employed for a more detailed insight into the results.

The execution of all methods on the three selected apps was successful. Based on the considerations for the visualization made in Section III-C, the results are shown in Figure 3. Again, the rows show the three different methods of investigation for each of the three specimen. The columns show the identified trackers (**DT4**) or permissions (**DT5**). Colored boxes indicate that a method identified the tracker or permission within the specific specimen.

In case of trackers no *intra-engine* differences between *EPO* and *ESA* are visible. There are some differences when compared to the findings of *APC*. *Chrome* contains two trackers (Google Ads and Facebook Ads) according to *APC*. *Fennec F-Droid* contains Mozilla Telemetry which *EPO* and *ESA* identify as a tracker. *Mozilla Firefox* has three trackers common to all methods. These are Facebook Ads, Google Firebase Analytics and LeanPlum. *APC* also identifies GoogleAds as a tracker. *EPO* and *ESA* identify GoogleAdMob instead. All three methods identify the same set of permissions.

Dynamic analysis can be used in order to verify these observations. The setup described in IV-A was used to execute *Fennec F-Droid* within a "low-noise" environment using *Wireshark* [11] to record the network traffic before reviewing said traffic in order to identify the data flows. In order to reduce the influence of background noise, *Fennec F-Droid* was started multiple times and the resulting communication behaviour observed. Every time the app started, a connection to firefox.settings.services.mozilla.com was attempted. Hence, we can conclude a DF_{tp}, since the host does not correspond to the provider of app. This confirms the result of the static analysis with regards to *Fennec F-Droid*.

Example **ES2** clearly shows the limitations of the approach presented here. The identification of trackers only yields DF_{tp}) and does not identify DF_{fp}). This leads to the good results of *Chrome* in this comparison without taking into account any DF_{fp}), which might cause privacy violations. Here, a thorough dynamic analysis might provide additional insight. Either

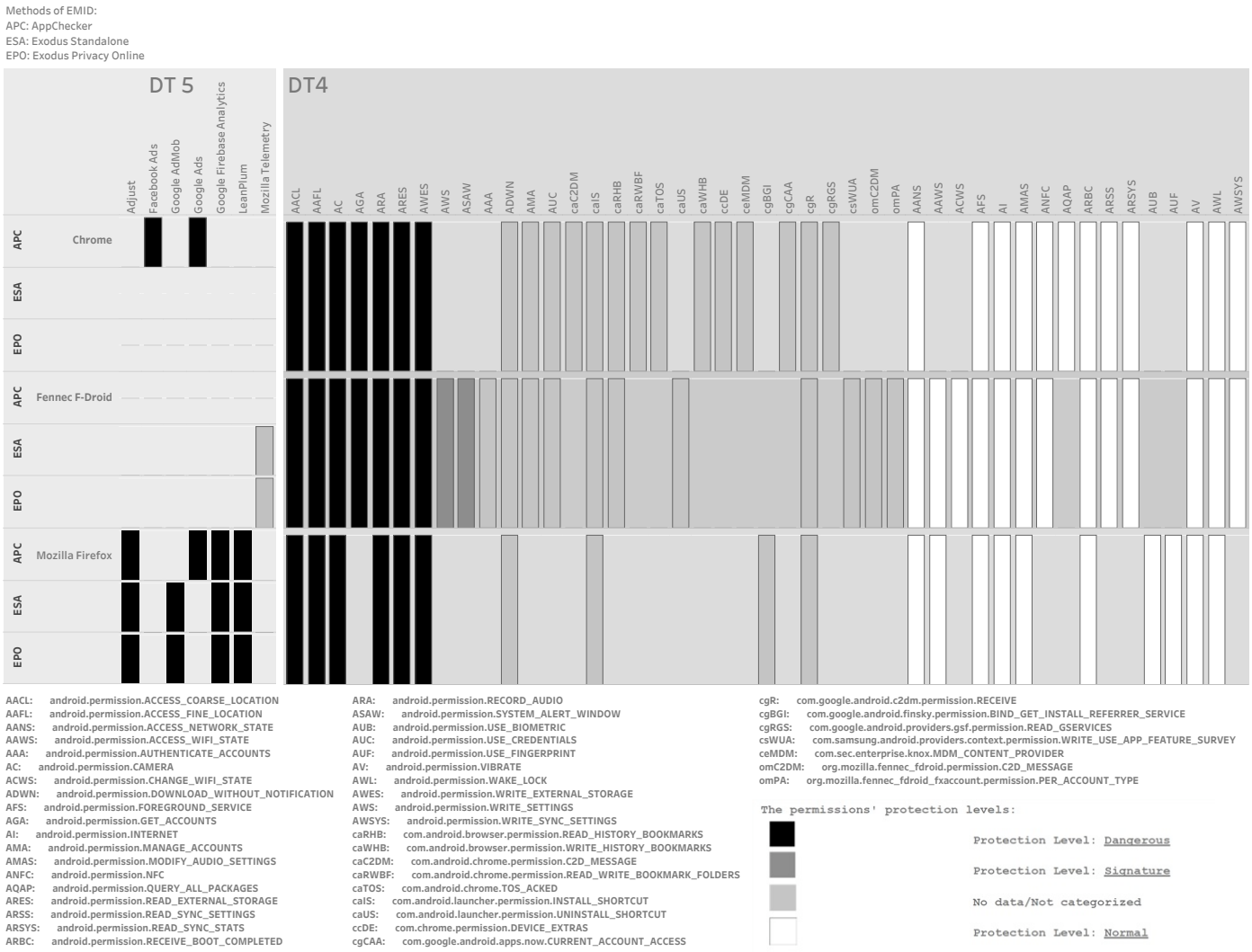

Figure 3. Trackers and permissions identified during **ES2** performed on three different web browser apps performed in Section IV-C

examining the specific data flow or blocking the specific data flow and then checking if the app still performs its function can be used to discern whether a data flows represents $DF_{fp.req}$ or $DF_{fp.nrq}$ respectively. This approach is also usable to separate $DF_{tp.req}$ from $DF_{tp.nrq}$.

V. Conclusion

This paper discusses potential privacy violations during the use of mobile apps caused by data flows to the first party and third parties. It introduces four different types of data flows with potential privacy implications and some means to detect these data flows. It discusses how the results originating from various methods can be made more reliable and more comparable. This is achieved by applying practices from computer forensics to this field and describing the examination processes from the viewpoint of a forensic investigation. This enables the identification of a set of relevant factors for selecting the methods of examination. In addition, considerations on a suitable visualization and an example for such an visualization are provided.

The approach is applied to two different Examination Scenario. In this first scenario, three different methods of static analysis to examine the potential data flows in a set of eight different apps are used. This leads to the identification of 42 trackers with 20 of them being different. In addition, 167 permissions are identified with 77 of them being different. The second scenario examines three different browser apps and identifies trackers and permissions. It shows the limits of the presented approach and gives a short example of a dynamic analysis.

The approach presented in this paper relies on static analysis and shows the limitations of this approach. The identification of trackers should use varied methods in order to achieve conclusive results since the methods presented here might provide differing results. In addition, they do not identify DF_{fp}. This would require additional methods. The advantage of the approach presented within this paper is that it provides comparability of the achieved results, which is not provided if only a single isolated method of examination is used.

The approach presented within this paper focuses on the identification of trackers from a purely technical standpoint and hence gives no guidance whether a specific tracker would violate the privacy of the user from a legal perspective. In

 ISBN: 978-1-61208-821-1

this technological view, the approach presented within this paper supports data sovereignty. Judging the impact a specific tracker has on the privacy of the user is a difficult task. Good guidelines are provided by the *Principles relating to processing of personal data* of Article 5 of the GDPR [1]. Chief among these is *data minimisation*. A negative example is the approach of marking tracking from certain providers as less critical since these providers already have access to similar data (like done in the grading scheme seen in [8]). If certain providers obtain personal data through the use of a specific tracker, they can use the personal data obtained through different trackers in order to create a profile.

The prevention of tracking is a complex topic. Users (including administrators or teachers in care of students) can try to block trackers using technical means. Also developers who might unwittingly include trackers in their software might take steps to remove these trackers. However, these methods are beyond the scope of this paper.

A. Future Work

The approach presented here deals with the identification of trackers (representing $DF_{tp.nrq}$) but also provides a foundation to examine DF_{fp}. This would include dynamic analysis and would entail blocking specific data flows and reviewing the apps functionality following. This remains an open topic as well as the potential inclusion of code review in order to improve results.

Additional future work should include research the provision of an environment that only captures the network traffic of a given app during dynamic analysis. At present, other processes can trigger traffic, creating false positives. In addition, a future extension could include the examination of websites using the same approach.

ACKNOWLEDGMENT

This document is partly funded by the European Union Project "CyberSec LSA_OVGU-AMSL".

REFERENCES

[1] European Union, "General Data Protection Regulation (GDPR)," 2020, https://gdpr.eu/article-5-how-to-process-personal-data/ [November 05. 2020].

[2] R. Binns, U. Lyngs, M. Van Kleek, J. Zhao, T. Libert, and N. Shadbolt, "Third party tracking in the mobile ecosystem," in Proceedings of the 10th ACM Conference on Web Science, ser. WebSci '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 23D31. [Online]. Available: https://doi.org/10.1145/3201064.3201089

[3] G. Maone, "NoScript," 2020, https://addons.mozilla.org/de/firefox/addon/noscript/ [November 05. 2020].

[4] pi-hole.net, "pi-hole," 2020, https://pi-hole.net/ [November 05. 2020].

[5] P. Jackson, "disconnectme - Tracker Descriptions," 2020, https://github.com/disconnectme/disconnect-tracking-protection/blob/master/descriptions.md [November 05. 2020].

[6] M. Maaß and D. Herrmann, "Privacyscore: Improving privacy and security via crowd-sourced benchmarks of websites," CoRR, vol. abs/1705.05139, 2017. [Online]. Available: http://arxiv.org/abs/1705.05139

[7] privacyscore.org, "PrivacyScore," 2020, https://privacyscore.org/ [November 05. 2020].

[8] mobilsicher.de, "So testen wir," 2020, **GERMAN** https://appcheck.mobilsicher.de/allgemein/so-testen-wir-schnelltest [November 05. 2020].

[9] android.com, "Android Operating System," 2020, https://www.android.com/ [November 05. 2020].

[10] Exodus Privacy, "Exodus Privacy," 2020, https://exodus-privacy.eu.org/en/ [November 05. 2020].

[11] wireshark.org, "Wireshark," 2020, https://www.wireshark.org/ [November 05. 2020].

[12] "A Road Map for Digital Forensic Research," DFRWS, Tech. Rep., 2001.

[13] R. Altschaffel, M. Hildebrandt, S. Kiltz, and J. Dittmann, "Digital Forensics in Industrial Control Systems," in Proceedings of 38th International Conference of Computer Safety, Reliability, and Security (Safecomp 2019). Springer Nature Switzerland, 2019, pp. 128–136.

[14] S. Kiltz, J. Dittmann, and C. Vielhauer, "Supporting Forensic Design - A Course Profile to Teach Forensics," in IMF '15: Proceedings of the 2015 Ninth International Conference on IT Security Incident Management & IT Forensics (imf 2015). IEEE, 2015.

[15] R. Altschaffel, K. Lamshöft, S. Kiltz, M. Hildebrandt, and J. Dittmann, "A Survey on Open Forensics in Embedded Systems of Systems," International Journal on Advances in Security, vol. 11, 2018, pp. 104–117.

[16] Exodus Privacy, "Exodus Standalone," 2020, https://github.com/Exodus-Privacy/exodus-standalone [November 05. 2020].

[17] J. Alemann and N.Baier and M.Streuber and T. D. Nam and L. Peters, "AppChecker," 2020, https://github.com/Tienisto/AppChecker [November 05. 2020].

[18] android.com, "Android Studio," 2020, https://developer.android.com/studio [November 05. 2020].

[19] genymotion.com, "GenyMotion," 2020, https://www.genymotion.com/ [November 05. 2020].

[20] ubuntu.com, "Ubuntu 18.04," 2020, https://releases.ubuntu.com/18.04/ [November 05. 2020].

[21] virtualbox.org, "VirtualBox," 2020, https://www.virtualbox.org/ [November 05. 2020].

[22] J. Palevich, "Android Terminal Emulator," 2020, https://play.google.com/store/apps/details?id=jackpal.androidterm& [November 05. 2020].

 ISBN: 978-1-61208-821-1