Christof Stromberger

Automated Network Communication Analysis of Mobile Applications

Master's Thesis

Graz University of Technology

Institute of Applied Information Processing and Communications
Head: Univ.-Prof. Dipl-Ing. Dr.techn. Reinhard Posch

Supervisor: Univ.-Prof. Dipl-Ing. Dr.techn. Reinhard Posch

Graz, April 2016

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, ________________________ ____________________________

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am ________________________ ____________________________

Datum Unterschrift

[1]Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Abstract

In a world of constant digitalization where almost every device is connected to the Internet, the data transferred between these devices and their servers becomes crucial. In fact, consumers spend most of their time on their mobile phones which raises the interest of advertisement and tracking providers. They are focusing on understanding and analyzing the behavior of the users in order to promote their products. For that reason, tracking and advertisement frameworks have been developed, which track down the user's preferences and actions. This fact raises concerns about the kind of data that is available to external providers, without that the user even realizes it. The main purpose of this thesis is to detect and understand the usage of such frameworks within mobile apps. Therefore, we investigate Android and iOS apps by automatically analyzing their network traffic. For our purposes, we collect the most downloaded apps from each category from the Apple AppStore and the Google PlayStore. Our main work is based on implementing a framework and developing five plugins that enable the user to automatically analyze the collected network traffic of the apps. Therefore, we collect the network traffic of the 440 apps by using a SSL Proxy. By using these plugins, we automatically observe the usage of unique device identifiers across multiple apps and different providers. Our results show that about 80% of all apps from both platforms are using at least one tracking or advertisement framework. Furthermore, we present that on average more than 50% of all apps are using the same unique device identifier in their network traffic, which enables advertisers to link the collected data and derive accurate personas from it. Moreover, we illustrate some security and privacy critical findings that are independent from tracking or advertisement libraries.

Contents

List of Figures

1 Introduction

Nowadays, we are living in a world of digitalization where almost every product or service can be digitalized and be represented by an app. While the usage of the apps is growing, the amount of apps in the app market is also growing significantly. The consumers spend most of their time on the phone in using apps [2] and therefore the advertisement providers are focusing on understanding and analyzing the behavior of the users in order to promote their products. For that reason, tracking and advertisement frameworks have been developed, which track down the user's preferences and actions. Through them, the behavioral data about the users is transferred over an Internet connection to the tracking providers. Thus, a huge amount of data collection about each user is stored in remote servers and is used afterwards for creating personalized advertisements that fit to each user's preferences. This fact raises concerns about the kind of data that is available to external providers, without that the user even realizes it. That leads to ethical and privacy related questions, that we need to ask ourselves. Are we as users aware of all the information collected about us? Is private and sensitive data also transferred to the external providers and what impact can that have to our lives? In this master thesis, we analyze the usage of tracking frameworks in 440 distinct apps, the usage of unique device identifiers that are used for identifying a particular device to the advertisement providers and the location of the destination of each network request that is sent from the analyzed apps. We try to raise some security awareness of all the data and information that we make available to third parties every time we are using an app.

The following section gives a background from which the interest and motivation of this master thesis has been evolved. Hence, it explains briefly the different kind of threats that exist for smartphone users and its possible implications.

1.1 Background and Motivation

Recent developments in the field of mobile security have led to a interest in privacy. For instance, sensitive data can be leaked through malware or in an indirect way via tracking and advertisement frameworks. As far as malware is concerned, it refers to many categories of malicious programs and depicts the same threat to a smartphone as it does to a computer. The term *Malware* includes adware, spyware, trojan horses, viruses, worms, keyloggers, ransomware and basically every application that might be harmful. Many of them contemplate to obtain sensitive data in order to use it for further attacks or for monetization. Hence, the unintentional or intended leakage of sensitive data can be caused by malware or malicious applications on the attacked device.

Nowadays, smartphones offer a variety of capabilities, such as location service, email messaging and third-party applications. Thus, huge amount of personal data is created and stored on each device. For instance, a regular smartphone stores location traces, contacts, photos, personal documents, call history, messages and much more [3]. Usually, this data serves a particular purpose, such as enhancing the user experience or offering various functionalities to the user. However, in recent years the so-called *personalized advertising* has been introduced. As a matter of fact, the more information is known about a specific user, the more personalized the advertisement becomes. Google describes it as a *targeted advertising* based on the interests and demographic characteristics of users [4].

However, targeted advertisements enable advertisers to serve personalized content that offers the highest possibility of a click. The more information they know about a particular user, the better they know which product is most likely to be bought by this user. Obviously, this is a conflict, since the advertisers are very interested in all kind of information that a smartphone can provide about the user. For that reason, common applications are used for collecting behavioral and demographic data. Concerning privacy, it is important to understand the implications that can occur when these data is shared with third parties and also combined together in order to create detailed user profiles. Common advertising frameworks not only provide the advertisement itself, but they also obtain data from the device and transfer it to their tracking servers [6]. Furthermore, it is possible that the app itself collects data which is then shared with so-called *ad networks*. This enables them to serve personalized advertisements based on the previously collected data. However, according to [5], such ad networks usually combine the data and sell it to the highest bidder.

Furthermore, cyber-criminals may want to collect sensitive information in order to commit identity theft or to steal money. Current smartphone generations store a huge amount of personal data that can be stolen and used for various kind of attacks. For this reason, it is advised to protect the user's device using a proper passcode. Another option that cyber-criminals use is malware, which is a popular way, since users tend to protect their smartphones less than they protect their computers [5]. Moreover, because of the increasing popularity of mobile payment solutions, where payment details are linked to a particular account, smartphones become a valuable target for cyber-criminals. Furthermore, geo-tagged photos offer the opportunity to determine where a particular user is working or when she is at home.

Additionally, the location and the activity of a person is an important component for law enforcement officers. Hence, persons of suspect who might have committed a crime may reveal important investigative information via their smartphones. Thus, the data stored on their device may be used in a court of law. It has been confirmed that law enforcement uses smartphone locations from carriers to create position traces [5]. Furthermore, the recent dispute between Apple and the FBI depicts that law enforcement agencies have even tried to break encrypted smartphones in order to gain evidence. Moreover, they have sought help also from the manufacturers and have proposed to implement backdoors in the encryption systems [7].

1.2 Privacy Concerns and Data Leakage

Considering the current development in the digital market, there is a probability that the usage of tracking and advertisement frameworks will increase. In fact, the smartphone market growths steadily since 2006. More than 1 billion devices have been sold for the very first time in 2014 and the market will reach 3 billion by 2020 [8]. This shows, that every year more and more people adopt to using smartphones. Moreover, this growth rate is backed by the emerging markets, such as Asia and Africa. On the other side, there is still a 25% share for feature phones in total mobile phone shipments, which will shrink by around 12% every year till 2020 [8]. Hence, that indicates a turn into fully adoption to smartphones in the next decade. Considering these numbers, there is a major interest in gaining more insight into the data that apps collect. The continuously increasing number of smartphone users elevates the significance of this analysis.

However, there are several mobile applications which transfer unintendedly the users personal information to advertisement servers by just using advertising frameworks [9]. Many developers are not aware of the privacy implications that may occur, when including such libraries. Moreover, these frameworks are usually compiled. Thus, it is not obvious for the developer what data is processed inside the library. Common tracking and advertisement frameworks use and transfer unique device identifiers such as the *Identifier for Vendors* or the *Advertisement Identifier* for uniquely identifying a person. Beside this, they transfer the location and other demographic attributes for gaining a more deep insight into the user. Apart from that, leak of sensitive data that is caused by permission violations is one of the main threat posed by malicious apps [10].

Lots of people still do not pay attention to the protection and privacy of their data. As a result, numerous of applications collect our data regarding the interaction between the users and the applications. Besides, frameworks that monitor the usage of the applications, the collection of technical information about our devices and much more, promise a more personalized user experience. But is this the real purpose of these applications or are they also used as a way of tracking, observing, analyzing and collecting behavioral data? The security and privacy is not obvious considering what happens when these data leak or when they are used in an unintended way.

1.3 Our Approach

The proposed framework and plugins in this thesis provide researchers and ordinary users with the ability to analyze the network communication of apps on their smartphones in an automated way. Its extendible architecture administers an easy way to analyze applications in order to detect suspicious behavior.

The main aspect covered by this master thesis is the ability to detect, monitor and analyze the transmission of user sensitive data through advertisement and tracking frameworks. Moreover, this thesis explores the big picture behind such frameworks and investigates the potential impact and consequences to the users when this tremendous amount of data from many apps is linked together. The secondary aspect is to enable the user to understand the transfer flow and to determine where the servers are located that process the data.

Therefore, five independent plugins have been developed that cover the usage of tracking frameworks and analyze such libraries. Furthermore, they monitor and analyze the usage of unique device identifiers that are used for linking the tracked data between multiple apps. In addition, another plugin that analyzes the network destination of each request and resolves the geo location. Furthermore, we implemented another plugin that visualizes the network traffic with all its requests and responses.

Before describing the plugins, the following sections define the terminology of advertising frameworks and the methodology of the proposed framework and its plugins.

1.3.1 Tracking and Advertising Frameworks

In the last few years there has been growing interest in mobile advertising, which made this ecosystem a powerful economic force. Furthermore, advertisers serve targeted ads based on the user's information collected by the apps [11]. In order to understand how such libraries work and why it is in their interest to collect data we have to discuss the different entities first.

As shown in Figure 1.1, such an ecosystem usually consists of **Advertisers**, optional **Agencies**, **Ad Networks** and **Users**.

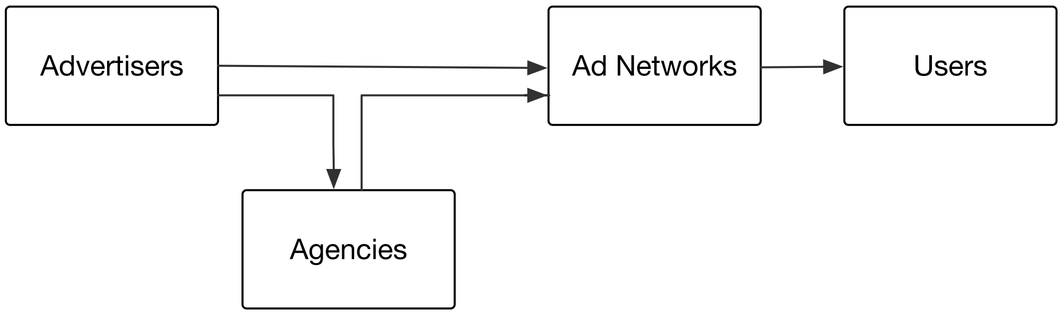


Figure 1.1: Typical Advertising Ecosystem

Originally, the mobile advertising techniques have been evolved from the methodologies that were used for web advertising. For instance, **Advertisers**, such as *Domino's*, *SEGA* or *foursquare* hire *Agencies* or provide directly *Ad Networks* with their advertisements. The **Agencies**, such as *McCann* and others distribute the ads to proper *Ad Networks*. This **Ad Networks** are companies that serve advertisements to the apps. Thus, they host the ads and provide an auction system to *Agencies* or *Advertisers*. In fact, such a network tracks the users' information in order to provide a personalized space for advertisers. Hence, many advertisers can bid on a particular space for an ad on a particular device and for an appropriate user profile. As a matter of fact, the more information the ad network can collect about their users, the more personalized the ads become. This technique has already been described in a patent from Google in 2001 [12].

This thesis defines *Tracking and Advertising Frameworks* as frameworks or libraries integrated in an app that collect personal, demographic, or technical information about a user. Certainly, only if collecting these data is not the main use case of the app itself.

1.3.2 Static and Dynamic Analysis

Before explaining the plugins, this section presents the methodology of the framework's architecture. As mentioned above, there are different kind of analyzing methods. The first technique is called **Static Analysis** and it is based on the program code and its execution flow. It is performed on a non-runtime environment whereby typically the program code is inspected and all possible paths that can occur during runtime are modeled. This draws attention to potential malicious behavior which typically is then analyzed by automated tools or manually from a researcher.

The opposite approach, that is used in this master thesis, is called **Dynamic Analysis**. It is executed during the runtime of an app and all necessary information are monitored and tracked. One advantage of using dynamic analysis is that the app is used like in a real world scenario. On the other side, it is difficult to reach all possible program execution paths. However, since a high code coverage was not necessary to find, understand and to analyze potential tracking libraries, we concluded to perform dynamic analysis.

Another dynamic approach is the **Taint Analysis**. In *Taint Analysis* the potential malicious data flow of applications is analyzed and presented to automated malware-detection tools or human analysts. It analyzes the recognized data leak and decides whether the leak actually is a policy violation or not. Thus, the sensitive data is tracked throughout the entire app. In order to track the data it is required to modify it by adding a kind of a tracking identifier. Hence, the tracked data can be distinguished from the regular data. A particular starting point, for instance a library call that provides the current location of the user, is needed for tracking a particular data object. The taint analysis tracks this data object through the data flow until it is used in a particular function that could leak it to an attacker, such as a network request. Taint analysis allows a statical and dynamical inspection of the app [13]. Though, dynamic app analysis requires many test runs in order to reach a sufficient code coverage. Furthermore, according to [13], current malware can automatically recognize dynamic monitors which may lead to wrong results since the app can hide its malicious behavior.

As opposed to dynamic analysis, static analysis does not face these problems. However, because of program abstraction it might be more imprecise. In fact, because of the abstraction it is not possible evaluate every possible path in the execution flow. The abstraction of runtime execution is even more challenging for mobile apps since they are integrated in the operating system's frameworks. Furthermore, apps use unpredictable callbacks or delegates for interacting with frameworks. Thus, for predicting the control flow, the entire lifecycle of an app, including GPS sensor input and user interface interaction, must be modeled.

1.3.3 Plugins

This section describes briefly the implemented plugins that were integrated into the proposed framework. First, we explain the *Tracking Framework List* and the *Tracking Framework Detail* plugins that are intended to detect and analyze tracking and advertisement frameworks within the apps. Then, we describe the *Device Identifier* plugin that detects unique device identifiers inside the network traffic and is able to link the usage of equal identifiers across multiple apps. Lastly, we explain briefly the *Location Destination* and *Timeline* plugins.

Tracking Framework List

The *Tracking Framework List* plugin analyzes a set of network dumps and detects if tracking or advertising frameworks have been used. Since the tool has only access to the network dump it detects libraries by checking the hosts of each request. Hence, if the requests is in a set of known hosts of tracking frameworks, it can be mapped and will be displayed as a positive detection.

Furthermore, it gives a brief overview about the results of the analysis by showing how many apps contained a tracking framework in percentage. Therefore, it states how many tracking libraries have been found in total within the analyzed apps.

In addition, it creates a ranking of the top 10 frameworks that were used in the set of analyzed apps. Thus, it monitors how often a particular library has been used in percentage of all apps that were analyzed. This enables the user to quickly understand which tracking libraries have been used most.

However, the main part is a detailed analysis of each application by monitoring how many hosts were actual tracking hosts within the network dump of a particular app. Furthermore, it monitors how much traffic (in bytes and percentage of the total traffic) has been sent to tracking servers. Furthermore, it states all hosts to which data has been sent in a ranked table. This enables the user to understand wether most of the traffic went to tracking hosts or to other ones. Thus, suspicious behavior can be detected more easily. Moreover, tracking hosts are highlighted in this table in order to see them instantly.

Tracking Framework Detail

This plugin analyzes the network dumps and if a request to a tracking or advertising host has been detected, it monitors the traffic that has been sent and performs an analysis of the data which has been sent. The plugin can detect 64 distinct tracking and advertising frameworks. Considering, that understanding the encoded traffic of tracking frameworks requires sophisticated manual analysis prior, the analyze method has only been implemented for some frameworks. But, due to the extensible architecture of the plugin, only one function must be implemented in order to fully integrate the analysis of a particular framework into the proposed framework of this thesis.

However, the plugin decodes and displays all the information that is sent to the tracking servers. For instance, the advertising library *Flurry* sends a unique device identifier, the device model, the screen height & width, the app identifier, the total amount of memory, the disk size that is used, the cpu load, the remaining battery of the device, the rough location where the device is located, which buttons have been clicked inside the app and much more.

Device Identifier

The plugin takes a set of network dumps from different apps and detects unique device identifiers inside the traffic. The same unique identifier is an indicator for a potential tracking across different apps because the data can be linked together on the servers. Thus, each request containing a particular unique device identifier is monitored and a report is displayed that shows all detected identifiers. Furthermore, it enables the user to see how many apps used the same identifier. For instance, in the *Games* category on the iOS platform 98 unique device identifiers could be found within 20 apps. Moreover, 17 out of this 20 apps contained exactly the same unique device identifier within their traffics. Furthermore, within this 20 apps 11 distinct tracking and advertising frameworks used the exactly same identifier for tracking the test device. This implies that not only these advertisers and ad networks can link data from different apps together, but even these companies could exchange the user's information between each other.

Considering, this tremendous amount of data from different apps out of different categories, such companies may be able to generate a highly accurate profile about a person.

Location-Destination

This plugin enables the user to gain insight to which country the data of an app is sent to. Thus, the country for each network request inside the traffic is determined by using a reverse lookup of the requests IP address. By analyzing an app that is supposed to serve a use case that is only needed inside Austria, we observed that it sends data to four other countries.

Timeline

We started by manually investigating network requests and figured out that a plugin is needed that displays this flow of requests and responses. Therefore, the *Timeline* plugin has been developed. It enables the researcher to monitor the network requests and the corresponding responses. It displays each request as a card on the left side with the related response on the right side. In order to enable the user to get a fast overview over the entire traffic we only display the host, the request method, a subset of the traffic's body and the HTTP response code. For further information about a particular request or response the details with all fields of the HTTP traffic can be shown. Secondarily, the plugin also shows the origin of the host by displaying a small country flag right next to it.

However, with this plugin it was easy to find potential attack vectors in API protocols. For instance, a widely used app on the iOS platform has sent the username and the password in plaintext over HTTP. This could be easily spotted by using the aforementioned plugin.

2 Related Work

Most research conducted on this topic focuses on Android devices and only few of them analyze the network traffic for tracking libraries. Hence, the contribution of this master thesis is an overall analysis across 440 Android and iOS apps focusing on security and privacy leakage of sensitive data through the network traffic. Moreover, most of the related papers use static, dynamic or taint analysis. Researchers have been studying this topic using different approaches which are presented in this section.

Before explaining the analyzing techniques, we categorize the related work by the different types of methodology. Considering *Static Analysis*, there are several solutions that have been proposed. For instance, M. Grace et al. have checked 100,000 Android apps for advertisement or tracking libraries in [9]. On the side of *Dynamic Analysis* researchers have been studying on the topic of sensitive data leakage and malware for several years in [10], [14], [11], [15] and [16]. Moreover, on the side of *Dynamic Taint Analysis* solutions that are integrated into the apps or the operating system and therefore need access to the code and the devices, are proposed in [13] and [17]. Lastly, the main topic of this thesis but with different approaches is discussed in papers [11] and [9].

To begin with static analysis and in terms of quantity, a large study about the usage of ad tracking frameworks in Android apps has been conducted by Grace et al [9]. They analyzed 100,000 apps from the official Android Market and concluded that at least one advertisement or tracking framework is embedded in more than 52% of the analyzed apps. Furthermore, they identified 100 representative ad libraries and developed a system that uses static analysis to categorize the danger of each framework. Basically, they scan the sampled libraries' sampled byte-code for dangerous API calls. After that, for each found call they slice backwards through the code searching for potential entry points. Their results show that most of the advertisement frameworks collect private information, such as call logs, phone numbers, browser bookmarks, or even the list of installed apps on the device. Moreover, they also found some frameworks that used *Reflection* to directly fetch and run code from remote servers.

Based on static analysis, the authors in [13] propose a static taint-analysis tool called *FlowDroid*. As they state, important data is either leaked by accident from careless developers or by malicious apps which exploit given permissions to copy data intentionally. According to Arzt [13] existing static analysis can show high number of false positives. FlowDroid is based on novel on-demand algorithms that yield high precision while maintaining acceptable performance [13] et al. It is developed for Android and analyzes the byte-code and configuration files of the app in order to find potential privacy leaks. According to the authors, *FlowDroid* is the first static taint-analysis that is fully sensitive for the context, flow, field and objects. It is designed to minimize the number of missed leaks by fully handling the entire Android lifecycle and tracking all callbacks and user interface interactions within the app. It can be used for securing and analyzing in-house developed Android apps as well as a tool for helping security researchers to inspect potential malicious Android apps. They released *FlowDroid* under open-source to provide it to the research community. Furthermore, they introduce DroidBench, an open-source micro-benchmark suite for evaluating the effectiveness and accuracy of Android based taint-analyses'. It is designed to provide a comparison for existing static taint-analysis tools. *FlowDroid* achieved 93% recall and 86% precision, which outperforms commercial tools, such as IBM AppScan Source and Fortify SCA [13]. It has found leaks in 500 apps from Google Play Store and in around 1,000 known malware apps from the VirusShare project.

Tracking libraries cannot only be detected with static analysis, but also using dynamic analysis, which is presented in the next papers. Chen et al [11] and the authors investigate the risk of privacy leakage through analytics services on mobile devices. They demonstrate how to extract individual profiles and usage information from external Google Mobile App Analytics and Flurry. Furthermore, they demonstrate how to attack such services by injecting arbitrary usage data into the analytics and advertisement services. Thus, the advertisements provided by the adversary and served to the users, are manipulated. As they state, analytic libraries and advertisement libraries collect usage information and user related attributes and send them to an aggregation server. Global unique identifiers that are available on mobile devices, such as the *iOS Advertisement Identifier* and Google's *Android Advertisement ID*, allow analytics services to connect data collected from different mobile applications to the same individuals. By using these data, the advertisement service can derive personas and categorize individuals. In that way, they can provide personalized advertisements.

The proposed model in [14] considers that malware often connects to remote servers in order to leak private information or is remotely controlled by some servers in order to get commands from them. They divided their framework into two phases. In the first they analyze the network traffic of known Android malware. They distinguish between malware and normal traffic. Using the difference in the traffic they can separate the malware traffic from the normal traffic by finding a list of features. This list considers *"Average Packet Size"*, *"Average Number of Bytes Sent per Second"*, and much more. In the second phase, they build a classifier based on decision trees since it is easy to interpret and classify new examples quickly. Using this rule-based classifier they derive three risk levels. According to these risk levels and the number of features they found in the traffic, they categorize each application according to its risk level. They consider 27 malware examples that have been obtained from the Android Malware Genome Project whereby only 13 malicious servers were online. Furthermore, the authors assume malicious behavior by looking at particular features in the traffic. According to their results, malware traffic can be distinguished from normal traffic by observing these features.

Another approach for finding data leakages has been conducted from Felt et al. in 2011. Although, they have also identified tracking libraries within the apps, they have given a different name to it. In particular, researchers from the University of California analyzed 46 pieces of malware, across iOS, Android and Symbian, in 2011. They found out that 61% of them collected user information and 52% were used for sending premium-rate text messages [10]. They used known malware collected from public anti-virus databases and discovered that many of them have used root exploits for gaining access to system functionality and data. Furthermore, they categorized the malware regarding three distinct threat types, which are

Malware, *Personal Spyware* and *Grayware*. In fact, their description of the latter is basically the same what we call *Advertisement & Tracking Libraries* nowadays and is also related to this thesis. Although this paper is relatively old, it points out the early stages of malware and describes the beginning of information tracking for advertisement and marketing purposes.

According to the authors in [10], there are several reasons why malware exists. They focus on some of them in detail, such as selling and stealing user information. Apps provide access to a large amount of information about their users by using the underlying functionality of the operating system. Sensitive data, such as geographical location, contacts, browser history and in particular cases unique device identifiers. According to the authors, 28 out of the 46 apps have collected these kind of information. They also mention that advertising or marketing companies might be interested in these data for using them for defining user profiles. Another scenario they mention is the theft of user credentials. Nowadays, people use smartphones for sensitive tasks, such as banking, shopping and more. Revealing such credentials can be a reasonable target of malicious apps. As of 2008, bank account credentials and credit card numbers have been worth $10 to $1,000 respectively $0.10 to $25 [10]. Furthermore, having installed malware on a user's device can enable attackers to initiate calls or text messages to premium rate numbers. Such actions can cost several dollars per message. However, in the 1990's such methods were used by desktop malware for financial gain and has now also been adopted to mobile apps [10].

On the side of manually performed dynamic analysis the paper of Schrittwieser [15] studies the authentication schemes of popular messaging clients on Android and iOS. All of these apps are based on an authentication scheme that uses the user's phone number for registration and a code via text messages for verification. The paper evaluates the security of mobile messaging apps and focuses on potential abuses in real-world scenarios, such as account hijacking, sender spoofing, message manipulation, unrequested messages, enumeration of existing users and the modification of status messages. In total, 9 apps have been analyzed. However, the probably most well-known apps of the analysis were *WhatsApp*, *Viber* and *Tango*. They found out that *WhatsApp* generated the verification code on the client side at that time. Thus, it is easy to hijack an account by intercepting the code when it is sent to the server instead of getting the code from the text message. When verifying this account, which is identified by the phone number, through the aforementioned code the attacker can authenticate an account successfully without being in possession of the actual phone number. Other apps, such as *Tango* do not require a phone number verification if the number has not been registered already. This allows an attacker to impersonate a user respectively his phone number. Other apps in the analysis do not have a verification at all. For instance, in one analyzed app, the phone number is linked to an account by just entering it [15].

The authors also describe the enumeration of existing accounts. Most analyzed apps offer the ability to automatically import the user's address book. In fact, all contacts are compared to already registered phone numbers on the server. Thus, the server returns a subset containing only the available users identified by their phone numbers. This functionality allows the identification of active phone numbers. In the paper they analyze all possible phone numbers of the southern part of the city of San Diego, California. There was no limitation in place, only a minimal slowdown of server response times. After 2.5 hours they obtained 21.095 valid phone numbers including their status messages [15].

The field of dynamic analysis has also been explored from Enck et al. and been extended by using a taint analysis for Android devices. Current smartphone operating systems provide control for permissions, which enable apps to access user related information, device information, data from sensors and much more. However, they do not provide insight into how private information is used by the apps. Therefore, in the paper [17] Enck et al. introduces *TaintDroid*, an Android extension which is capable of simultaneously tracking the flow of multiple sensitive data sources. It considers third-party apps as untrusted and therefore monitors and alerts the user in real-time whenever an app is trying to access or manipulate the users' private data. According to the authors, the main idea was to detect whenever sensitive data leaves the operating system in order to conduct further analysis on this particular app. Although this approach is not new, they have accomplished an appropriate balance between accuracy and performance. In addition, the authors' experiments have shown a performance overhead of 14% while running *TaintDroid*. Their evaluation was based on 30 randomly selected apps in which they found 58 instances of potential misuse of the users' private information. Moreover, 15 out of the 30 apps have reported the users' location to remote advertisement servers. The latter mentioned aspect is also covered in this thesis. In contrast to the paper, the captured network traffic is used for determining which data is sent. Hence, the analysis is not carried out on the device itself.

The authors in [16] introduce a different approach for privacy leakage detection based on models that analyze the users' intentions. In contrast to the proposed methodology of this thesis, they propose an automated app analysis and check if the data sent via network traffic is a privacy leak or an intended and expected behavior of the user. They motivate their approach with the fact that privacy leakage by mobile apps is a subject that needs reconsideration. For instance, the user's location can either be sent for a legitimate use case or can be unintentionally leaked. Thus, transmitting sensitive data per se, does not necessarily indicate a privacy leakage. They distinguish between *user-intended data transmission* and *unintended data transmission*. For instance, they presume that before sending a text message the user should click on the touchscreen and therefore this transmission is considered as legitimate and intended. Moreover, they point out that usually when the user's location is sent to a server, interesting contents tailored to the location are returned. This kind of pattern complies to the user's intention and should not be treated as a privacy leakage. On contrary, private data transferred without the user's interaction or when it is irrelevant to the core functionality of the app, is considered as an unintended data transmission. As a result, the latter mentioned scenario happens in a stealthy manner.

However, due to the complex relation between data leakage and user intention it is almost impossible to have an automated analysis. Therefore, they provide a human analyst with an automated tool to have all the contextual information in which the potential data leakage happens. As a consequence, they have developed *AppIntent*, which tracks user interaction inputs and sensitive data transmissions for showing context information of the network transfer in form of a sequence of user interface manipulations. They have analyzed 750 reported malicious apps and 1,000 free apps from the Google Play Store. Based on this, they present two case studies from their analysis which show one app with user-intended transmission and one without user's intention. In the latter, a malicious app stealthily sends the user's location to third-party servers [16].

Compared to this thesis the authors have chosen a similar approach but they have used an automated instead of a manual process for collecting the app's network traffic. Moreover, their focus had been on data leakages and not the effects and influences of accessing sensitive user data across multiple apps.

3 Background Knowledge

For understanding the methodology and evaluation a particular terminology is used, which requires some background knowledge. Therefore, this chapter gives a brief overview about methodology, tools and principles that are used in this thesis. However, researchers and analysts who are well versed in the field of IT-Security and specifically in mobile platforms and possible attack scenarios may skip this chapter.

This chapter is separated into three parts. In the first part security related background knowledge is explained. The methodology in this thesis refers to these principles and methods. Secondly, privacy topics are presented, which are used in the evaluation and in the results chapters. Lastly, the different app distribution techniques on iOS and Android are briefly described.

3.1 SSL Proxy

In general, a proxy is used for routing network traffic through a particular point. This point is usually a server or device that receives the traffic and forwards it to its final destination. However, the *SSL Proxy* is a specific structure, which acts like an additional end-point for both sides. For the client, the *SSL Proxy* operates like a server. Therefore, it connects and authenticates to the real remote server, which is the expected final destination of the request. Then, it generates a new certificate and replaces the original public key with a known one. Moreover, the original issuer of the certificate is replaced by its own certificate authority. Hence, it is necessary to trust this root certificate on the client. This usually happens by adding the certificate to the client's trust store or trusted certificates. When the client accepts the injected certificate, it sends a shared key, which is encrypted by using the certificates public key, to the server. Because of the aforementioned replacement of the actual encryption key, the *SSL Proxy* can decrypt the shared key and is able to eavesdrop the communication [18].

In figure 3.1 the different states are shown and afterwards it is explained how a *SSL Proxy* is used for eavesdropping a HTTPS secured connection.

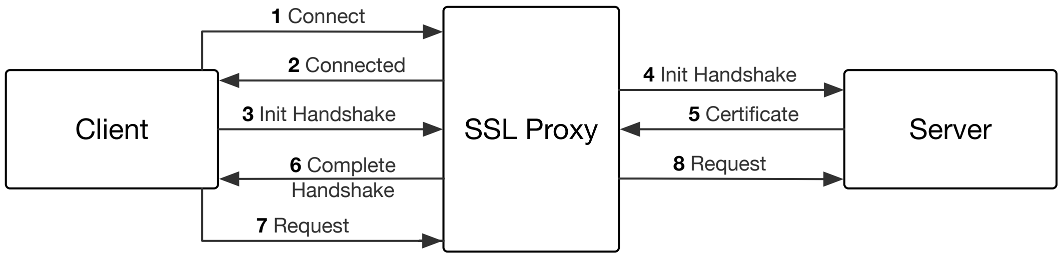


Figure 3.1: SSL Proxy with certificate replacement [1]

Initially, in state **1** the client connects to the server via *"HTTP CONNECT"*. Secondly, the *SSL Proxy* returns *"HTTP 200: OK"* in state **2**, which informs the client that the connection has been successfully established. Afterwards, in the third step (**3**), the client initiates the SSL handshake and assumes it is connected to the remote server. However, it is connecting to the proxy. Then, it is using the *Server Name Indication (SNI)* to express the hostname to which it is trying to connect. In state **4**, the proxy connects to the actual remote server by initiating an SSL handshake. It establishes an SSL connection by using the aforementioned SNI that has been obtained from the client. Afterwards, in step **5**, the server checks the transmitted SNI and returns a matching SSL certificate. The certificate contains the *Common Name (CN)*, which is crucial for the *SSL Proxy* for generating the replacement certificate. In the next step (**6**), the replacement certificate is created by the proxy. Then, the SSL handshake phase is terminated that has been initiated by the client in step **3**. In fact, the SSL connection through the proxy has now been established and the client is ready for communicating with the remote server. Thus, in step **7**, the client sends the request to the remote server over the proxy like a regular transmission. Last of all, the proxy forwards the clients request from state **7** to the server by using the SSL connection preluded in step **4**. [1]

3.2 Certificate Pinning

Transport Layer Security (TLS) has been introduced to secure protocols in the application layer. Furthermore, it is used for securing virtual private networks (VPN). However, a critical point in establishing a secure connection is the authentication and the key exchange. This is usually performed by using *X.509* certificates. An attacker could inject her certificate, which would enable a *Man in the Middle Attack*, described in Section 3.3. Hence, an attacker can eavesdrop, intercept and also insert traffic in the network communication. This hinders confidentiality and integrity and therefore, leverages the purpose of TLS [19]. In order to establish the required trust in the exchanged certificate, *Public Key Infrastructures*, so called PKIs, are used. Therefore, a certificate authority (CA) is considered as trusted and all certificates issued by this authority are also trusted. The client validates the received certificate by validating the so called chain of trust. Thereby, signed certificates are traced back, up to a trusted CA. In the Internet, this method is the common standard and is widely considered as secure [19].

The security relies on the trust established with a certificate authority. One of the purposes for introducing TLS with certificates from PKIs was to encrypt the communication and therefore, to prevent *Man in the Middle Attacks*. Every trusted *Certificate Authority* can issue trusted certificates for any domain name. The security can be thwarted accidentally or intentionally from a trusted CA by replacing a valid certificate with a malicious one [19].

For this reason, *Certificate Pinning* has been developed. The client verifies the authenticity of the server's public key by pinning a trusted and known certificate. Therefore, the client must know the associated host and the pin. This requires information about the certificate and the host on the client. Thus, existing applications must be modified and extended to be capable of this action. However, a client can detect any change of the hosts certificate or public key and deny the connection [19].

3.3 Man in the Middle Attack

In general, a *Man in the Middle Attack* (or *MITM* attack for short) can be conducted in various situations. In fact, the official definition in the RFC2828 is the following: *"A form of active wiretapping attack in which the attacker intercepts and selectively modifies communicated data in order to masquerade as one or more of the entities involved in a communication association"* [20]. Thus, the communication between two parties is targeted. Hence, client-server communication are also affected by this kind of attack. According to [21], SSL/TLS authentication is performed by the user and usually done in a poorly or naive way. Another critical point is the aforementioned *SSL Proxy*, described in section 3.1. For these reasons, a MITM attack is considered as a realistic attack scenario in the context of this thesis. However, we focus on attacks within SSL secured communications and declare other MITM attack scenarios as out of scope for this section.

TLS has been introduced for protecting client-server applications from *Man in the Middle Attacks*. However, one key factor of the systems' security is the authentication and the key exchange. This is usually performed by using X.509 certificates. Considering a vulnerability in the key exchange, opens an attack vector for a third party. This attacker would not only be able to eavesdrop the communication between the client and the server, but it would also be able to intercept and insert traffic. Hence, an altered SSL certificate could be injected, like it is described in section 3.1. In fact, the attacker would act like a person in the middle that can decrypt and eavesdrop the communication between two parties. This is why it is called *Man in the Middle Attack*. [19]

Figure 3.2 illustrates a *Man in the Middle Attack*. Basically, an attacker was injected between the client and the server and is able to eavesdrop the communication.

In a typical example for a MITM attack, the attacker injects itself between the client and the server in a way that allows to communicate separately to both parties. In fact, neither the user on the client, nor the server is aware of the third party in the middle. SSL encrypted network traffic can be decrypted by the attacker and re-encrypted, using a known public key, for the other party. For instance, assume that a user tries to login into her bank account.

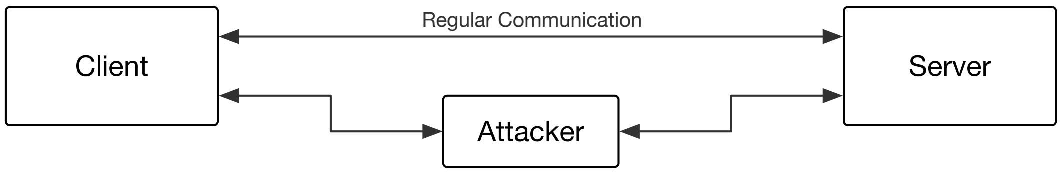


Figure 3.2: Man in the Middle Attack

Then the attacker can grab the password or the security token in order to authenticate himself and impersonate the user. [21]

3.4 Unique Device Identifiers

Service providers and app developers need to uniquely identify a device for legitimate reasons. However, for advertisement providers it is necessary to uniquely track devices, which in fact, belong to a particular user. This implies that identifying a device also would mean to identify a user. In fact, advertisement libraries need these identifiers to track the users' behaviors for providing them with more personalized advertisements. The so-called *UDID* on iOS, made it possible to link collected data from multiple apps. In other words, advertisement companies could generate a profile about the user and his or her habits across different apps. This practice raised some privacy concerns in the past, because they are considered as a more personal information than, for instance, cookies [22]. Both major mobile platforms, iOS and Android, offer such a unique device identifier.

As privacy concerns raised, Apple deprecated the access to the devices identifier with the release of iOS7 in 2013. Basically, they replaced it with two different identifiers: the **Identifier for Vendor** and the **Advertising Identifier**. The former identifier is the same for apps that were distributed and uploaded to the AppStore from the same vendor [23]. The identifier remains the same while the app is installed on the device. It is only deleted, when the user has removed all apps from the same vendor from the device. Moreover, when such an app is reinstalled, the identifier for vendors changes. [23]

The second identifier, which has been introduced on iOS as a replacement for the *UDID*, is the **Advertising Identifier**. It is unique to the device and available to all apps that use *iAd* for advertising [24]. Unlike the *Identifier for Vendor*, the same id is returned to all apps even from different vendors. However, it can change when the user resets the device or it can be reset manually by the user in the settings of the phone. The intention behind this was to limit advertisement tracking for the user. [25]

Google uses a similar approach on Android. Although there is still the *Android ID*, which is a unique identifier generated at the first startup of the device, they enforce the app developers, who deploy user tracking to use the so-called *Advertising ID*. Unlike in iOS, the former mentioned unique device identifier can still be used. [26][27]

3.5 Permission System on Android and iOS

Both platforms, iOS and Android, are based on a permission system for granting access to device functions and user information. In general, both provide apps with an extensive API that covers access to settings, user data and hardware, such as gyroscopical sensors or GPS. On **Android**, access is granted on an install-time basis. Thus, once the user has downloaded the app and accepted the permissions, the app can access all the requested functionality. In Android 6.0, permissions that are requested while the app is running, have been introduced. At the time of writing only 2.3% of all android devices used Android 6.0 [28]. Hence, the wide-spread and primarily used install-time approach will be exposed in this section.

According to [29], the Android SDK provides 235 possible permissions in total. However, per default, apps are running in a low-privileged process and can only access their own files. Moreover, each application is deployed on its own virtual machine. The permissions are categorized into *Normal*, *Dangerous* and *Signature/System*. The first describes functionality that cannot harm the user. For instance, *SET_WALLPAPER* is such a *normal*-permission. *Dangerous* permissions contain potential harmful functions, such as accessing sensitive user data, or sending text messages. The last category (*SignatureSystem*) controls access to the most harmful API calls. In fact, it includes functionality, which is related to the operating system itself, such as deleting or installing apps. However, these permissions are only granted to apps that were signed with the manufacturer's certificate. These apps are usually pre-installed on the device. In general, some API calls are protected by permissions. For instance, whenever an app tries to access the user's contacts, it must have the *READ_CONTACTS* permission allowed.

By comparison, Apple's **iOS** uses in-app permissions, ever since they have introduced their permission system. The apps are isolated, which prevents them from accessing files from other apps. Moreover, unauthorized changes to the device and the operating system settings are prevented. A randomly generated and unique home directory is created for every installed application. Within this directory, all the files of the app can be saved. The underlying API calls restrict access to the app by using particular permissions for certain functions. Moreover, the permissions are based on entitlements, which are set in the app's provisioning profile. Restricted functionality that is enabled in this area and has been approved by the user can be accessed by the app during runtime. Furthermore, these entitlements cannot be modified because they have been signed by the developer's certificate which is issued by Apple. [30]

Unlike older Android versions, the iOS user has the ability not only to accept all permissions at install-time, but also to accept single permissions at runtime whenever the functionality is accessed by the app. Furthermore, given permissions can also be revoked later on, under the settings of the operating system. This approach gives to the user full control over apps permissions.

3.6 Apple App Store

The App Store from Apple has been evolved from the iTunes Music Store, which has been initially launched in 2003. Although, the App Store is based on the same principle, it distributes mobile apps for iOS and not music, as iTunes does. It allows users to browse and search for apps in the Apple ecosystem. In fact, it provides more than 1.5 million apps and has reached the 100 billion downloads mark in 2015 [31][32].

Apple uses a manual review process for approving or rejecting apps. The reviewers mainly look for stability, user interface inconsistencies and usage of undocumented API calls [33]. Furthermore, malware will be rejected because of Apple's approval guidelines. Thus, going unnoticed through this process can be challenging for developers of malicious apps. After approval, the app gets signed by Apple and is executable on any iOS device. One drawback of this approach is the time, since an app review usually lasts about one week and can be shortened by asking for an expedite review [33]. However, this is only allowed in case of time critical issues. In order to communicate all limitations and review rules, a comprehensive document called review-guidelines has been published by Apple for all iOS developers in 2010.

3.7 Google Play Store

The *Play Store*, also known as *Google Play*, was previously named *Android Market*. It follows the same principle as Apple's *App Store* for distributing apps developed for Android. It has been launched in 2012 and in 2015, there were already over 1.43 million published apps [31].

Google introduced the so-called *Google Bouncer* that checks submitted apps for potential viruses. It automatically analyzes, detects and removes malware from the store [34]. Interestingly, from 2011 to 2013 the number of malicious apps increased by 388%, whereby the percentage of the removed apps decreased from 60% to 23% in the same period of time [35]. However, the approval process is based on automated checks and apps that violate the store guidelines are removed.

4 Core Framework

The base framework is the underlying foundation for the developed plugins described in chapter 5. It is based on the *Play Framework*[1] which is an open-source framework for web applications written in Java and Scala. It uses the MVC-Pattern[2] and provides several components for rapid prototyping, such as *RESTful* webservices and *Object-Relation* mappers. Thus, it has been used for this thesis in order to provide a fast and stable foundation in which the plugins for the analysis have been integrated. Java is used for the main part of the framework which includes the entire business logic described in Section 4.2 whereby the frontend part, using the Play template engine, is written in Scala. This part is describes in Section 4.1.

However, the framework is intended to upload, process and analyze network traffic dumps from *Burp Suite*. This tool is used for collecting the network traffic and is briefly described in Section 6.2.1. In fact, the proposed framework is written in a generic way for an easy extension for other network traffic capture formats, such as *Wireshark*[3]. Therefore, only a parser for this has to be developed.

In order to understand the main functionality provided to the user we will describe the frontend part in the next section.

[1]www.playframework.com

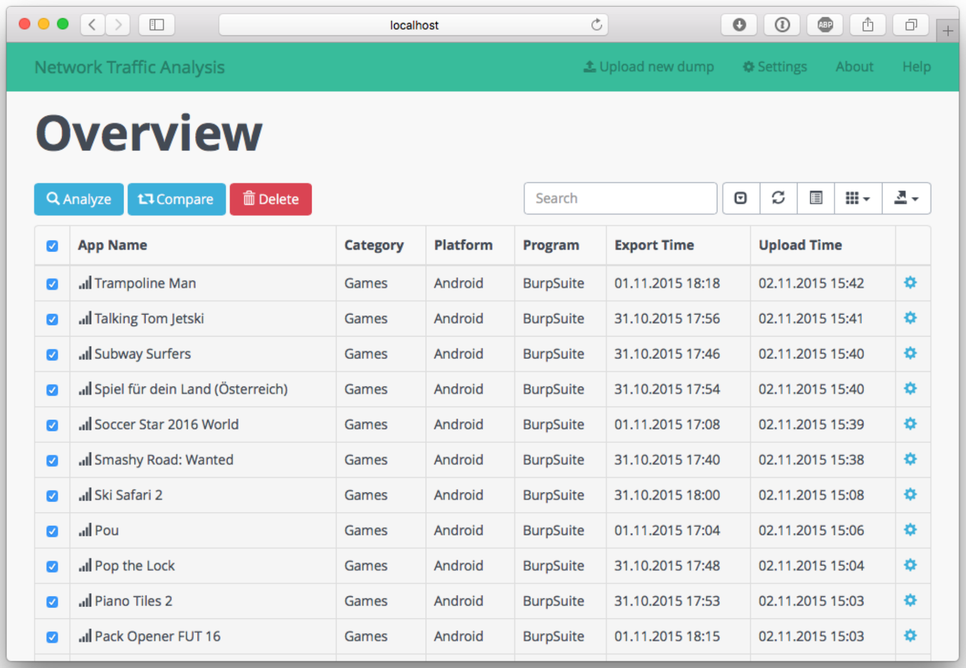
[2]The **M**odel-**V**iew-**C**ontroller pattern is an architectural pattern in software design. It divides an application into separate layers for storing, processing and displaying information.

[3]https://www.wireshark.org

4.1 Web Frontend of the Core Framework

The core framework uses *Scala HTML* for presenting the user interface. Since the usability of the web application is not a topic of this thesis, we will just briefly describe the main functionality. Figure 4.1 shows the home page that lists the uploaded network dumps and provides the functionality for uploading, analyzing and the selection of plugins.

However, the user can select one or all network dumps from the list shown in the center of Figure 4.1. After selecting, the analysis can be started by pressing the *Analyze* button. Then, the selected plugins perform their analysis on the aforementioned dumps. In order to insert collected network traffic into the underlying *Elastic Search* database, which is described in Section 4.3, the web application offers an upload method that can be triggered by clicking *Upload new dump*. The user can enter the *App Name*, *Version*, *Platform*, *Category* and if the app is a free or a paid version.



Overview

☑	App Name	Category	Platform	Program	Export Time	Upload Time	
☑	Trampoline Man	Games	Android	BurpSuite	01.11.2015 18:18	02.11.2015 15:42	⚙
☑	Talking Tom Jetski	Games	Android	BurpSuite	31.10.2015 17:56	02.11.2015 15:41	⚙
☑	Subway Surfers	Games	Android	BurpSuite	31.10.2015 17:46	02.11.2015 15:40	⚙
☑	Spiel für dein Land (Österreich)	Games	Android	BurpSuite	31.10.2015 17:54	02.11.2015 15:40	⚙
☑	Soccer Star 2016 World	Games	Android	BurpSuite	01.11.2015 17:08	02.11.2015 15:39	⚙
☑	Smashy Road: Wanted	Games	Android	BurpSuite	31.10.2015 17:40	02.11.2015 15:38	⚙
☑	Ski Safari 2	Games	Android	BurpSuite	31.10.2015 18:00	02.11.2015 15:08	⚙
☑	Pou	Games	Android	BurpSuite	01.11.2015 17:04	02.11.2015 15:06	⚙
☑	Pop the Lock	Games	Android	BurpSuite	31.10.2015 17:48	02.11.2015 15:04	⚙
☑	Piano Tiles 2	Games	Android	BurpSuite	31.10.2015 17:53	02.11.2015 15:03	⚙
☑	Pack Opener FUT 16	Games	Android	BurpSuite	01.11.2015 18:15	02.11.2015 15:03	⚙

Figure 4.1: Screenshot of the web frontend of the core framework

4.2 Framework Architecture

The framework is split up into 5 different parts, which are *Controllers*, *Model*, *Plugins*, *Util* and *Views*. This structure is also shown in Figure 4.2. The purpose of the controllers package is to provide a central point for interaction between the views and the actual implementation. Furthermore, the upload of data dumps and the plugin registration is handled in there.

The core framework is implemented using a *RESTful* webservice for providing an interface between the user interface and the actual plugins. In particular, Play uses a *routes*-file that specifies the API. Basically, it is the entry point for network requests in the web application. The responsibility of the controllers is to handle the requests and return the expected views

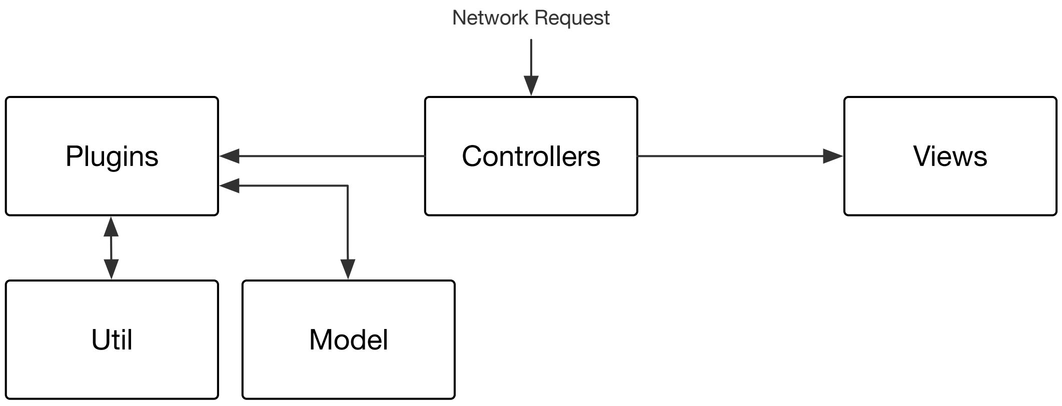


Figure 4.2: Structure Overview of the Core Framework

or data. The controller part, that is also shown in the figure 4.3, can be structured into the following pieces.

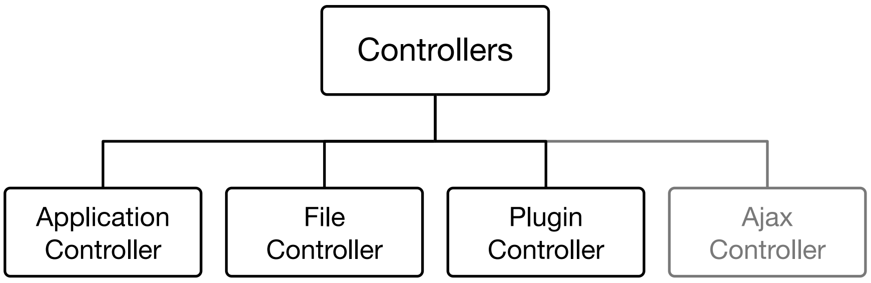


Figure 4.3: Controllers in the Core Framework

ApplicationController It handles all application related requests and is used as the main interface for the web application. It only consists of `GET`-Requests and provides the browser with the necessary HTML user interface for working with the application. It provides the `/overview` API, which displays the overview of all collected apps that are in the database. Furthermore, it offers an `/upload` functionality that shows the upload form for collecting new network dumps. This has been implemented in order to upload new network traffic dumps to the server and establish a set of apps that can be analyzed. Moreover, under `/dump/show` a specific network dump for a given identifier can be shown. In addition, it offers a `/search` functionality for finding particular network dumps that contain a specified query string. However, the main purpose of the *ApplicationController* is to provide the `/dumps/analyze` method. This takes a given set of network dumps as a parameter and invokes the analysis method of the plugins on this set. The enable and disable functionality of a plugin is handled by the *PluginController*.

FileController The second controller is responsible for managing the network dumps. Furthermore, it interacts with the *Elastic Search* database, described in section 4.3. The *FileController* provides the `/file/uploadFiles` functionality, which is the business logic behind the `/upload` form described in the controller above. It handles the incoming requests, which contain the network dumps and meta data, such as *App Name*, *Platform* and *Category*. Then, it stores everything into the database. By now, the controller only supports dumps exported by *BurpSuite*. However, this can be extended easily in the future. Furthermore, it offers `/file/deleteDump` and `/file/deleteDumps` functionality, which are responsible for deleting a particular network dump or respectively a set of dumps.

PluginController Thirdly, the *PluginController* enables the user to activate or deactivate a particular plugin. It provides the functions `/plugin/enable` and `/plugin/disable`. The user can enable or disable plugins in a checkbox before invoking the actual analyze functionality, described in the *ApplicationController*. Due to the fact that this thesis is based on several plugins that are integrated in the core framework, the *PluginController* enables the user to activate or deactivate particular analyzing methods. For instance, by controlling which plugin is enabled, the user can change the focus from privacy analysis to data leakage detection and much more.

AjaxController Lastly, the *AjaxController* only provides the `/ajax/getLocation` function. This method is used in some plugins for resolving an IP address regarding the geographical location of the network request. Often it is interesting to know the country of the host. Thus, this method has been implemented. However, a geographical search based on lookup servers that are able to resolve most IP addresses, may be time consuming. Hence, this functionality has been provided using *Ajax* technology.

The plugins controlled by the *PluginController* perform their analysis on the data within the uploaded network dump. Each dump contains the traffic from one session of an app. All this data is stored in the underlying database. The next section describes briefly the basics of *Elastic Search* that is used in the proposed framework as data storage.

4.3 Elastic Search

The proposed framework uses Elastic Search as data storage. Due to its high flexibility and out-of-the-box functionality it was suitable for this case of application. In order to understand the concepts behind Elastic Search, this section gives a brief introduction.

Elastic Search is basically a search engine based on a database. It has been published in 2010 by Shay Banon and became one of the most downloaded open source projects within a few years [36]. It uses so-called **Indices**, which basically represents a particular part in the file system where cohesive data is stored. An Index is like a database in a SQL server. The search engine reads and stores data or entire documents in the selected index. In fact, Elastic Search uses the Apache Lucene library for accessing and storing data in an index.

In addition to indices it offers so-called **Documents** that include one or more fields. Data is stored as a document, which can be described as an entity that is used for representing the data. The search engine uses these documents for searching. As mentioned before, each document consists of fields, whereby each field has a name and a value. Furthermore, the value itself can also be an array of one or more fields. For instance, a document can be a JSON object that is passed to the index and stored there.

Another term is the **Mapping**. Every document is analyzed before storing and it can be specified what should happen with the data, such as removing HTML tags, before the document is stored. In addition, Elastic Search can automatically detect the type of the field based on its value.

Each document has also a specified **Type**, which is like a table in SQL. This allows to store different kind of documents with different structures.

Another important feature of Elastic Search are **Nodes**. Based on Lucene it is implemented in a highly scalable way. A single server or instance is called a node. For this thesis a single-server node was sufficient but it can also be extended with more nodes which is called a cluster then.

Such a **Cluster** is a set of independent nodes that work together. The load and requests are managed and divided to all nodes in the cluster. Another advantage is that some nodes can be maintained while the entire system works as expected. Elastic Search can store more data than a single server could do.

This is achieved by using so-called **Shards**. For the user it appears as a huge single index but in fact the data is split up into shards that can be spread over all available nodes. All of this is handled automatically by Elastic Search.

Because of this distributed system, they introduced **Replicas**, which are are small parts of the shard that are copied and distributed over all nodes. For instance, if the node with the shard is switched off, Elastic Search can recover all data and information from the other nodes. [36]

In the proposed framework integrates Elastic Search via the provided REST API. Basically, JSON encoded HTTP requests are sent to a particular index endpoint of the API. Therefore, a document is sent to a particular index. For instance, in this thesis one index is used for all the data dumps that were collected. For retrieving data from the API, the so-called *Query DSL Language* has been introduced by Elastic Search. It enables the user to send simple or complex queries for requesting documents from one index. For more information on *Query DSL*, please read the documentation in [37].

The network dumps that are stored in *Elastic Search* are collected separately with *Burp Suite*, which is described in detail in section 6.2.1.

5 Plugins

We started by investigating the network traffic sent by mobile applications on iOS and Android. During these observations we encountered several significant issues that we analyzed manually. Thus, we developed automated plugins for the prior implemented core framework in order to automate our analysis. Due to the extendable architecture of the proposed framework, we were able to extend it at any direction that our observations pointed out. Basically, the input for every plugin is the captured network traffic. It is dumped by using *Burp Suite*, which is described further in Section 6.2 *Experimental Setup*.

Our main focus was to develop plugins that help security researchers and regular users to easily understand and evaluate what happens with their data when it leaves their devices. Thus, we have developed the following plugins: *Analysis of Tracking Frameworks*, *Detailes Analysis of Tracking Frameworks*, *Usage of Unique Device Identifiers*, *Network Destination Location* and *Request-Response Timeline Visualization*.

Hence, four out of five plugins focus on the topic of what happens when the data is sent via Internet to unknown servers. One plugin has been developed in order to visualize the timeline of HTTP requests and responses. This enables the user to skim through the network dumps easily and also enables her to manually analyze the sent traffic.

The findings in the manual analysis, which has been conducted by us prior writing this thesis, as also the recent concerns about the *Safe Harbor* agreement between the U.S. and the European Union, motivated us to analyze the data for privacy related issues. The other four plugins cover the topics of tracking and advertisement frameworks, usage of so-called unique device identifiers and analysis of the network destination locations.

This chapter is split up into five sections whereby the first four cover the aforementioned topics and the last section describes the timeline visualization plugin.

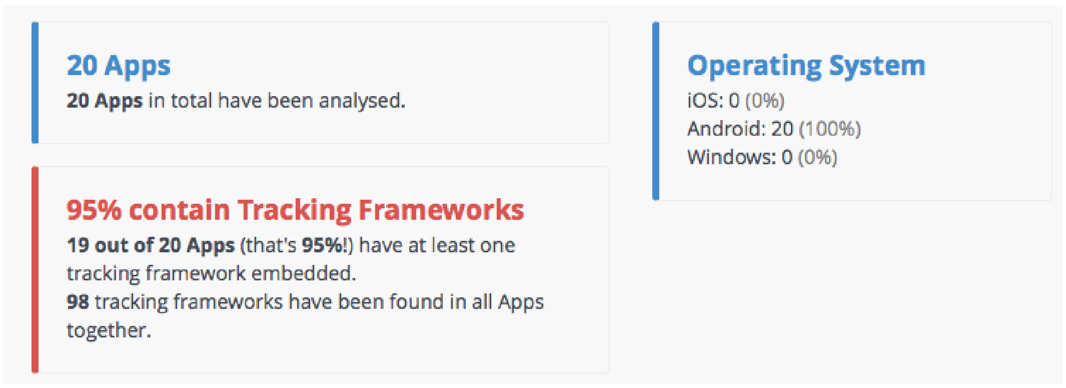
5.1 Analysis of Tracking Frameworks

One of the key findings in the manual network traffic analysis that has been conducted prior writing this thesis, was the usage of tracking and advertisement frameworks within the traffic. A sheer amount of network requests were targeted to tracking servers, not to legitimate webservices that serve the actual purpose of that app. For this reason, it was required to gain more insight and give a fast overview about all the metrics necessary for security researchers in order to understand and evaluate the network traffic for the usage of tracking libraries. Thus, we implemented this plugin that analyzes all network requests and categorizes them according to wether it was a tracking host or not. It further helps the user to quickly check the percentage of used frameworks, and much more.

However, from all the analyzed data a brief overview is created, which indicates how many apps have been used in at least one tracking library and how many have been found in total. The plugin also analyzes on which operating system the app was running so that the user can compare different platforms. A brief overview of this plugin is illustrated in the following Figure 5.1.

In order to create the aforementioned summary and for analyzing the network dumps, the plugin uses the *Framework Manager* for loading the registered frameworks and making use of them. We developed this generic framework manager in order to enable developers to easily extend these plugins. The structure of this is illustrated in Figure 5.2.

Currently, the plugins can detect 64 known and widely used tracking and advertisement frameworks. All of them transfer data to their hosts. However, the framework classes are registered by the manager automatically through *Java Reflections*. In this way the plugin can be easily extended by creating a new class in the right package and inheriting from *BaseFramework*. Then,



20 Apps

20 Apps in total have been analysed.

95% contain Tracking Frameworks

19 out of 20 Apps (that's **95%**!) have at least one tracking framework embedded.

98 tracking frameworks have been found in all Apps together.

Operating System

iOS: 0 (0%)

Android: 20 (100%)

Windows: 0 (0%)

Figure 5.1: Screenshot of the tracking framework plugins overview

the next time the application is executed, the plugin can detect the newly added tracking framework and use it within its analysis.

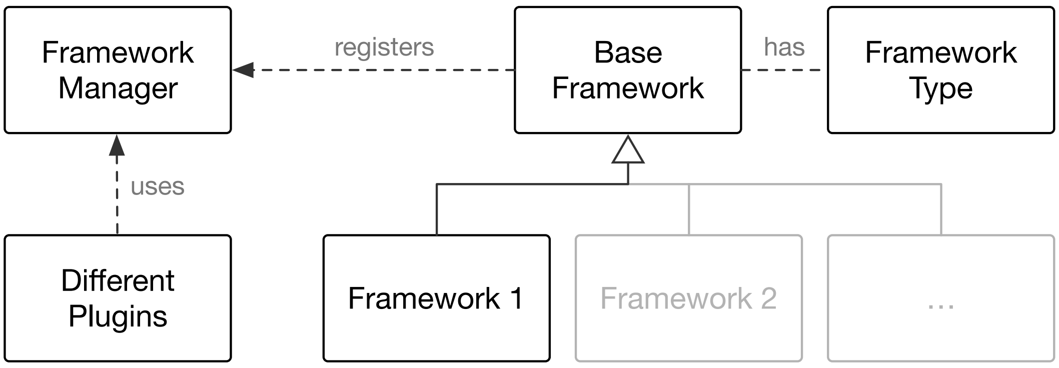
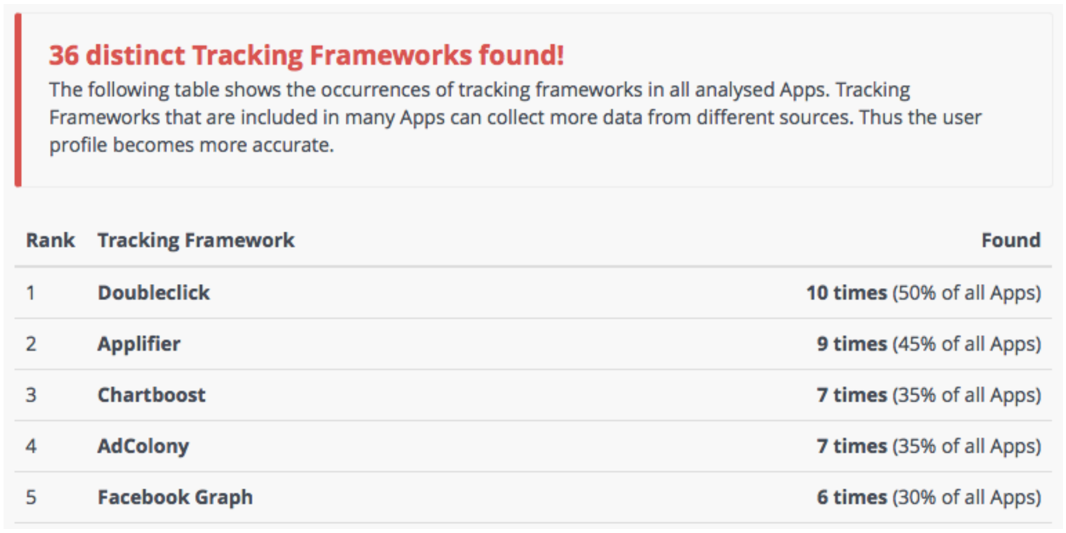


Figure 5.2: Structure of the *Framework Manager* and its *Frameworks*

The plugin analyzes the network dump of every selected app. For each detected framework or library it monitors the traffic that has been sent to their hosts. After finishing the analysis, the plugin shows the top 10 libraries within all selected mobile apps ranked by their usage. In fact, the detection is based on a host comparison. Therefore, the plugin checks the host of every outgoing network request within the captured traffic and compares it with the known tracking hosts of all registered libraries. The frameworks are provided by the aforementioned *Framework Manager* and every library can contain multiple hosts. Whenever a match is detected, this request and all its data is treated as a potential finding. For instance, a sample output for the top 20 Android apps in the *Games* category is shown in Figure 5.3.



36 distinct Tracking Frameworks found!

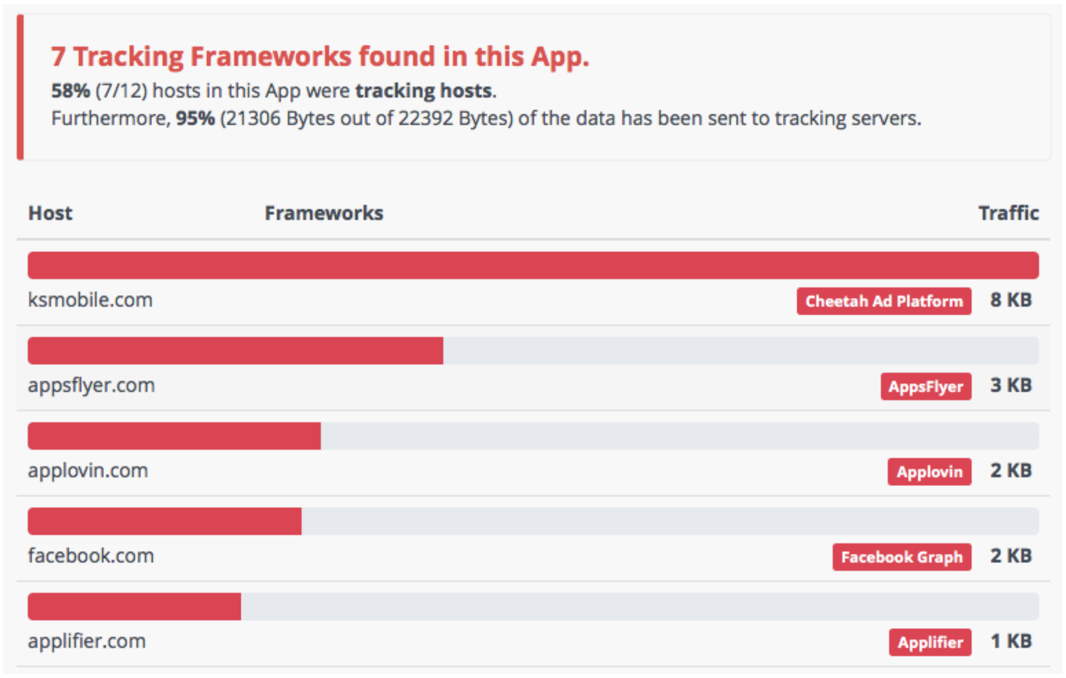
The following table shows the occurrences of tracking frameworks in all analysed Apps. Tracking Frameworks that are included in many Apps can collect more data from different sources. Thus the user profile becomes more accurate.

Rank	Tracking Framework	Found
1	**Doubleclick**	**10 times** (50% of all Apps)
2	**Applifier**	**9 times** (45% of all Apps)
3	**Chartboost**	**7 times** (35% of all Apps)
4	**AdColony**	**7 times** (35% of all Apps)
5	**Facebook Graph**	**6 times** (30% of all Apps)

Figure 5.3: Screenshot of the top 10 framework analysis created by this plugin.

In addition, it provides information regarding the usage of each app. As already mentioned, this plugin works on per app basis. Thus, every app that has been selected by the user in the frontend of the core framework, is analyzed individually. The aforementioned overview and the top-ten statistics combine these results. The main purpose of this plugin is to easily understand and analyze if a particular app is suspicious or not. Furthermore, the number of frameworks and the percentage of data that is sent to tracking hosts, can be used as an indicator for evaluation.

However, the plugin summarizes how much traffic in bytes has been sent to each host, including the servers from the tracking and advertisement libraries. An example from the Android Games category is shown in Figure 5.4. The app sends 95% of its traffic to potential tracking hosts.



7 Tracking Frameworks found in this App.

58% (7/12) hosts in this App were **tracking hosts**.
Furthermore, **95%** (21306 Bytes out of 22392 Bytes) of the data has been sent to tracking servers.

Host	Frameworks	Traffic
ksmobile.com	Cheetah Ad Platform	8 KB
appsflyer.com	AppsFlyer	3 KB
applovin.com	Applovin	2 KB
facebook.com	Facebook Graph	2 KB
applifier.com	Applifier	1 KB

Figure 5.4: Screenshot of a detailed analysis of a particular app by this plugin

5.2 Detailed Analysis of Tracking Libraries

The manual analysis which has been conducted prior writing this thesis, has shown that tracking frameworks or libraries send user specific data to their hosts in order to gain more information and insight into their users. Hence, the network communication between the app and the tracking hosts is analyzed within this plugin. The plugin also monitors if sensitive and privacy related data, such as the device name or uniquely identifiable device information, is sent by tracking or advertisement libraries. Especially in the latter case, it can be crucial because the advertisers can connect user-specific data throughout other apps. Furthermore, many advertisers obtain sensitive user information in order to create a more transparent persona for serving better personalized advertisements. Consequently, it is possible in that way to create better targeted campaigns as part of their business.

Unfortunately, each library transmits data in a different way. Therefore, we implemented an automatic decoding and analyzing function for selected libraries. This plugin has been developed in order to analyze and understand the data that such libraries really send to their servers. It is based on the framework classes and the *Framework Manager* described in the previous section and shown in Figure 5.2. Hence, it uses the same classes as the previous plugin but extends them with an *analyze* functionality. Whenever this method is implemented in the framework class, it automatically analyzes the network dump and presents the result to the user. It was not in the scope of this thesis to implement the *analyze* function for every detected framework since it requires a manual analysis in order to understand the library. Thus, in order to verify the validity of the plugin, we implemented the analyze method for selected libraries, based on their occurrence in the manually performed analysis, that has been conducted prior writing this thesis.

Each recorded framework is implemented in a separate class. Therefore, it contains information about the library and it optionally implements the analyze method. We generated a class for every detected library. Thus, the framework classes are loaded via *Java Reflection* and can be extended for new frameworks during runtime. Every framework class inherits from the so-called *Base Framework*, which is initially described in Figure 5.2 and outlined further in Figure 5.5.

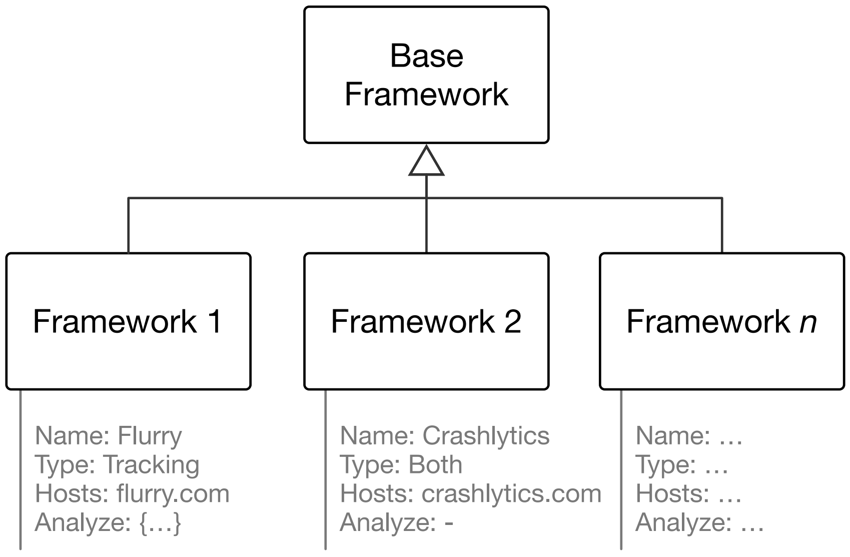


Figure 5.5: Structure of the framework classes

In fact, each framework is identified by several properties, such as **Name**, **Type**, **Hosts** and **URL**. For instance, in Figure 5.5 two distinct tracking frameworks are illustrated. The *Framework 1* implements the analyze functionality, whereby *Framework 2* does not. Considering that both libraries are detected by the plugin, the network dump affecting *Framework 1* is analyzed in detail and presented to the user. Considering the large number of detected libraries, not all frameworks implement this analyze method. However, it is easy to extend this work to all the available libraries.

The screenshot, shown in Figure 5.6 gives a brief overview about how many and which frameworks have been detected by the plugin. Therefore, the diagram illustrates the findings and enables the user to show the name of the detected library. Below the diagram the name of the application is displayed. Furthermore, it enables the user to investigate the data which has been sent by a particular library.

One challenge in implementing the *analyze* functionality was to understand and decode the traffic. Therefore, a manual process was necessary to under-

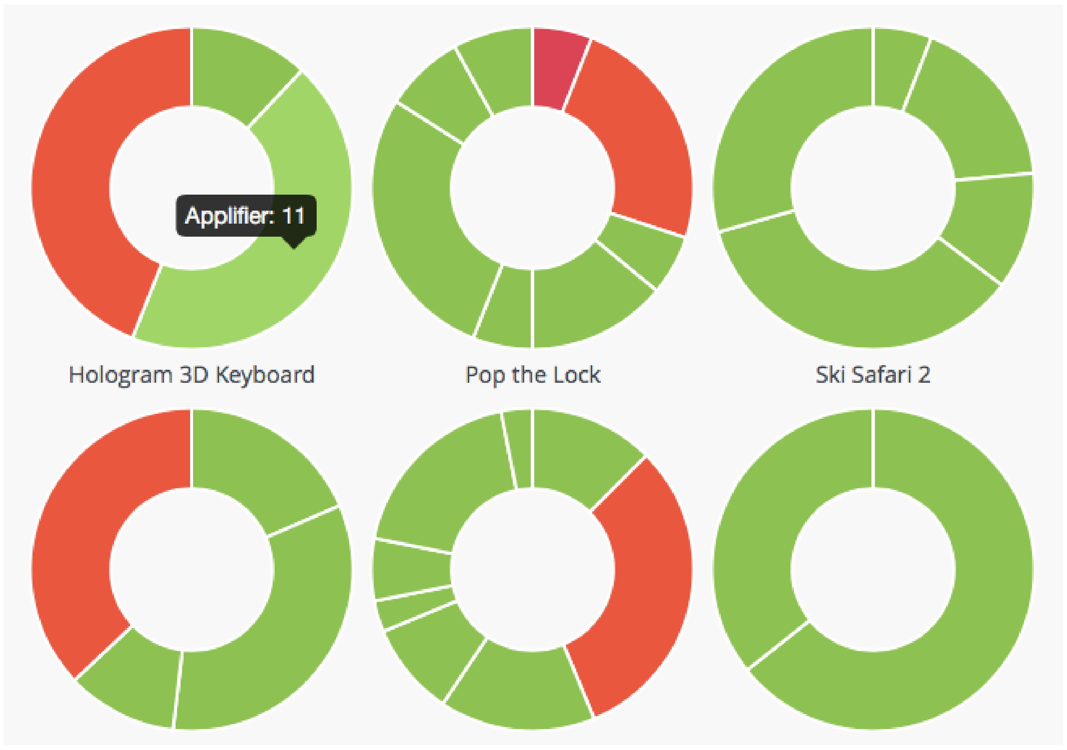
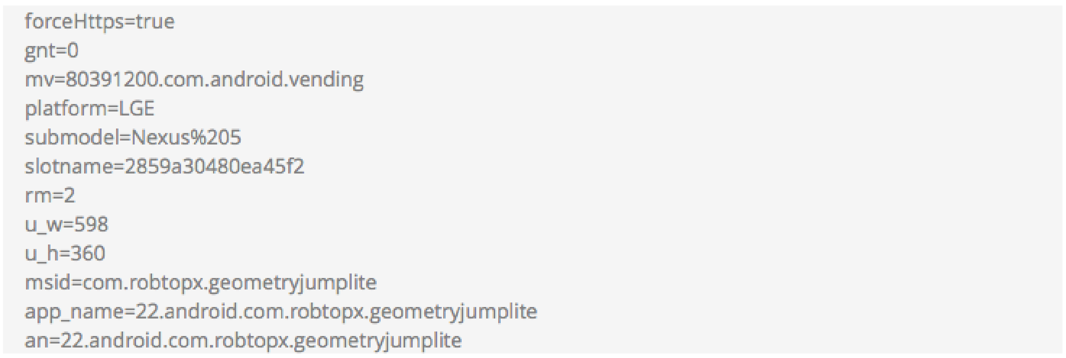


Figure 5.6: Screenshot of the *Detailed Analysis of Tracking Frameworks* plugin.

stand and decode the mostly obfuscated data. We overcame it by analyzing the body of the network request. We were able to determine particular patterns such as bit masks or semicolon separated values. The next screenshot, shown in Figure 5.7, shows the decoded network traffic, which has been sent from a tracking library to its host.



```
forceHttps=true
gnt=0
mv=80391200.com.android.vending
platform=LGE
submodel=Nexus%205
slotname=2859a30480ea45f2
rm=2
u_w=598
u_h=360
msid=com.robtopx.geometryjumplite
app_name=22.android.com.robtopx.geometryjumplite
an=22.android.com.robtopx.geometryjumplite
```

Figure 5.7: Screenshot of the decoded and analyzed data which has been sent from a tracking framework to its host.

5.3 Usage of Unique Device Identifiers

Most mobile apps utilize tracking or advertisement libraries. In fact, common libraries provide a benefit to the developer, by either monitoring the usage statistics of the app or by providing more personalized advertisement to the user. According to wide-spread advertisements and placement providers, they can increase the sales of an app by including their libraries into it [38][39]. Our analysis has shown that these providers might achieve this by collecting personalized data about the user in order to provide more accurate advertisements.

During our manual analysis prior writing this thesis, we constructed a hypothesis how advertisement providers could possibly link data sets from different apps together. Furthermore, linked data from different apps can improve the personalization significantly and major advertisement providers claim to be able to interconnect the gathered information [40]. Moreover, Flurry[1], one of these tracking and advertisements providers, keeps track of 1.4 billion devices across 540,000 apps [40]. This is one third of the entire app activity.

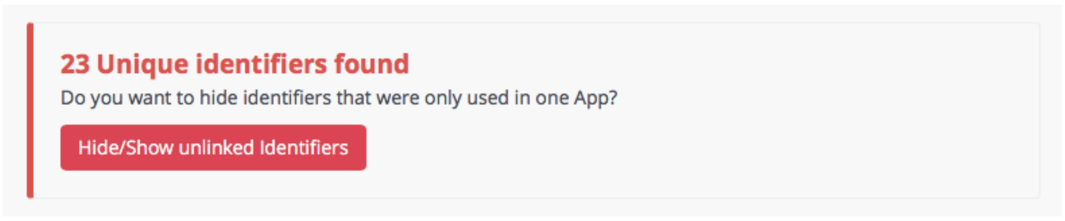
The main problem resulting from this functionality is that this sheer amount of data might be sensitive. Hence, it might have privacy and security implications. In fact, not only the age and the gender of a person can be derived,

[1]http://www.flurry.com

but distinct personas can be created from this data [40]. All this data and information about the user is identified by a single unique identifier. Hence, we developed this plugin in order to analyze the monitored network traffic and we searched for this identifier inside the network dump.

First, we created a plugin that enables the user to understand how many unique device identifiers have been found inside the analyzed network traffic. Considering that these unique identifiers are sent to the tracking servers, it enables them to bind all the data to one particular entity. Furthermore, we distinguish between *linked* and *unlinked* identifiers. A unique identifier that is accessed from at least two different tracking libraries is considered as *linked*. This implies that the same identifier is used by more than one provider. In fact, it enables them to connect the data, exchange it with other providers and generate a more accurate persona.

However, the so-called *Advertising Identifier* introduced by Apple and Google was intended to prevent the linking between different providers and apps. Our plugin enables the user to detect if her personal unique identifier is tracked by multiple frameworks or multiple apps. Hence, it analyzes if the tracked information about the user is linked together and how many providers can enrich this persona with additional data.

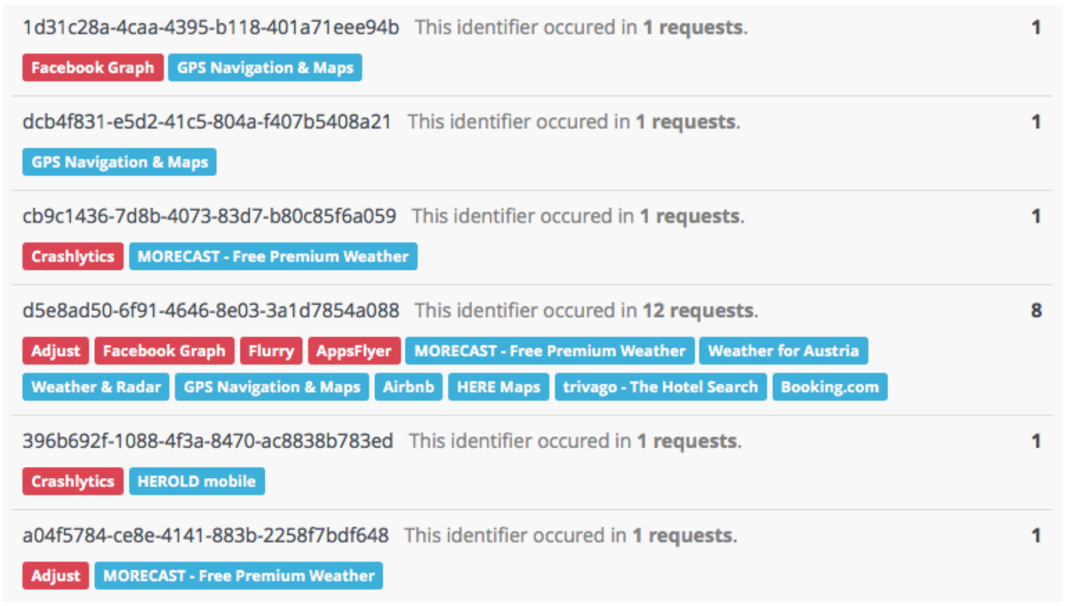

23 Unique identifiers found

Do you want to hide identifiers that were only used in one App?

Figure 5.8: Screenshot of Device Identifier plugin which scans for Unique Device Identifiers, so-called UDIDs.

Furthermore, the plugin shows the exact identifier that has been detected and displays a statistic about the number of requests in which this identifier occurred and within which frameworks and apps it has been found. In the conducted manually analysis prior writing this thesis we gained evidence that every tracking framework tries to uniquely identify a particular user. Thus, it is important to analyse the usage of the device identifiers. The following screenshot shown in Figure 5.9, displays the aforementioned statistic.

It presents the detected device identifier in the first row. Furthermore, it states how often this identifier occurred in different network requests. In order to understand which frameworks and apps used this particular identifier, the frameworks are shown as red labels and the apps as blue labels below the device identifier. The number on the right side states how many distinct apps used the exactly same identifier for transferring data.


1d31c28a-4caa-4395-b118-401a71eee94b This identifier occured in **1 requests**. **1**
Facebook Graph | GPS Navigation & Maps

dcb4f831-e5d2-41c5-804a-f407b5408a21 This identifier occured in **1 requests**. **1**
GPS Navigation & Maps

cb9c1436-7d8b-4073-83d7-b80c85f6a059 This identifier occured in **1 requests**. **1**
Crashlytics | MORECAST - Free Premium Weather

d5e8ad50-6f91-4646-8e03-3a1d7854a088 This identifier occured in **12 requests**. **8**
Adjust | Facebook Graph | Flurry | AppsFlyer | MORECAST - Free Premium Weather | Weather for Austria | Weather & Radar | GPS Navigation & Maps | Airbnb | HERE Maps | trivago - The Hotel Search | Booking.com

396b692f-1088-4f3a-8470-ac8838b783ed This identifier occured in **1 requests**. **1**
Crashlytics | HEROLD mobile

a04f5784-ce8e-4141-883b-2258f7bdf648 This identifier occured in **1 requests**. **1**
Adjust | MORECAST - Free Premium Weather

Figure 5.9: Sample statistic of the usage of device identifiers carried out by the analysis of the Device Identifier plugin.

In fact, the plugin has been developed to check wether the same unique device identifiers are used by frameworks across multiple apps. Furthermore, it points out that advertisement providers are able to exchange personal data in order to generate more accurate personas. Figure 5.10 depicts the structure. For instance, *Provider A* collects data from *App 1*, *2* and *3*. *Provider B* is only integrated in *App 1* and *2*. All the data which is gathered by the providers is linked to the same unique device identifier. Hence, the providers are able to exchange the connected data among themselves. Thus, *Provider B* can get the personalized data from *App 3* although it is not integrated in this app.

From a technical perspective, the plugin analyzes each request inside the

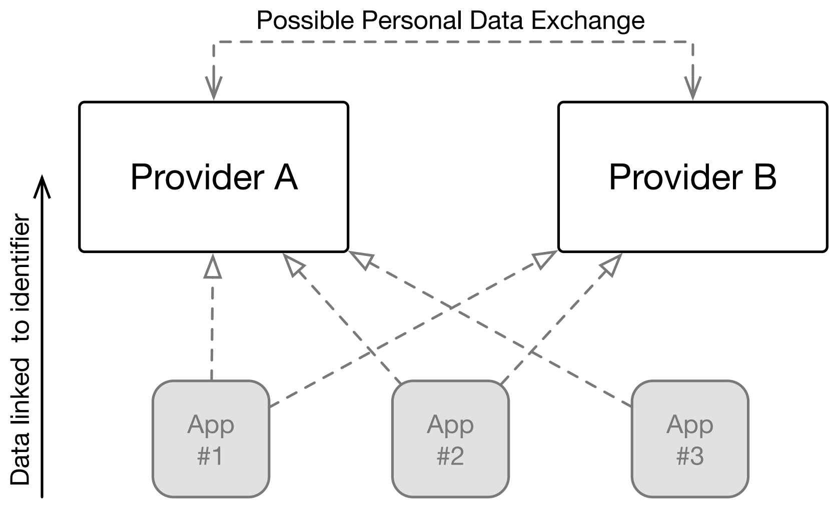


Figure 5.10: Structure that is used by providers and possible impacts of linked unique device identifiers.

network dump and checks with *Regular Expressions* if a device identifier pattern has been found. After that, it revises if the request was targeted to a tracking provider by integrating the functionality of the *Analysis of Tracking Frameworks* plugin, described in Section 5.1. If both match, the plugin considers the data included in this request as uniquely identifiable and tracked. This automatic analysis is performed for every network dump of the chosen apps. Thus, the plugin accumulates the data and detects if the same device identifier is used in different apps. In that way, it enables the user to understand which data, collected from different apps, might be linked together on the provider's side.

The plugin only checks for unencrypted, unhashed and unobfuscated identifiers inside the SSL encrypted network traffic. We decided, that monitoring the usage of obfuscated identifiers was considered out of scope and is open for future work. However, the main purpose of this plugin was to detect standard identifiers and to analyse if the providers are theoretically able to link the transferred data from different apps.

5.4 Network Destination Location

Most applications communicate with servers via the Internet for transferring data. In our manual analysis prior writing this thesis we found out that many servers are hosted in the United States. This implies that also the tracked data is stored there. Thus, this plugin considers the importance of the final destination of network requests. Moreover, privacy laws differ in many countries. Therefore, it becomes more and more important where, in particular in which country, the data is stored. The geographical location of the host address of each request is analyzed in this plugin. In order to give a brief overview about the variant locations, the data is visualized on a world map by using the Google Maps library. This is shown in Figure 5.11. Basically, the more requests are sent to a particular country, the more colorful the country becomes on the map. Hence, this option enables the user to easily identify which app should be analyzed further based on the number of requests per country.

However, we cannot determine what happens with the data after it has been delivered to the server. This aspect is considered as out of scope for this plugin.

Technically, the plugin uses a local webservice that is based on the Free-GeoIP[2] project in order to resolve the IP addresses. The project is inside the framework folder and is used as an API for querying countries based on hosts. The plugin implements the *LocationLookup* class, which opens a HTTP connection to the locally running GeoIP server. After that, it forwards the host's IP address to the *LocationLookup* and resolves the country location. In order to increase its performance, the plugin uses a memory cache for temporarily storing already checked IP addresses. Considering, that almost every network dump contains hundreds of requests to the same tracking host, the caching was a necessary step. Before the GeoIP server is queried, the network destination plugin checks the cache if it already contains the requested address.

In addition, the plugin caches the results. Whenever a particular network dump has been analyzed, the results are stored in an in-memory cache

[2]Locally deployed GeoIP Server used for resolving IP addresses: http://freegeoip.net

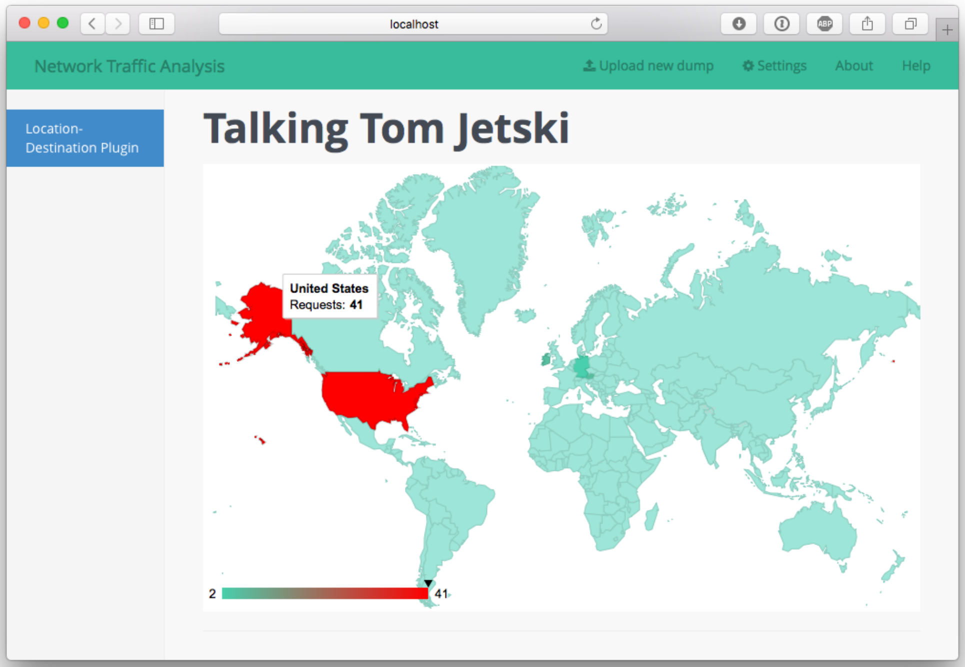


Figure 5.11: Screenshot of the plugin's results for a single analyzed application.

for further requests. Moreover, it caches the results of a network dump and not of the entire analysis. Thus, it is possible to run the plugin with more than one selected application. Hence, when the user analyzes one of these applications alone, the result will be instantly available because of the previous caching.

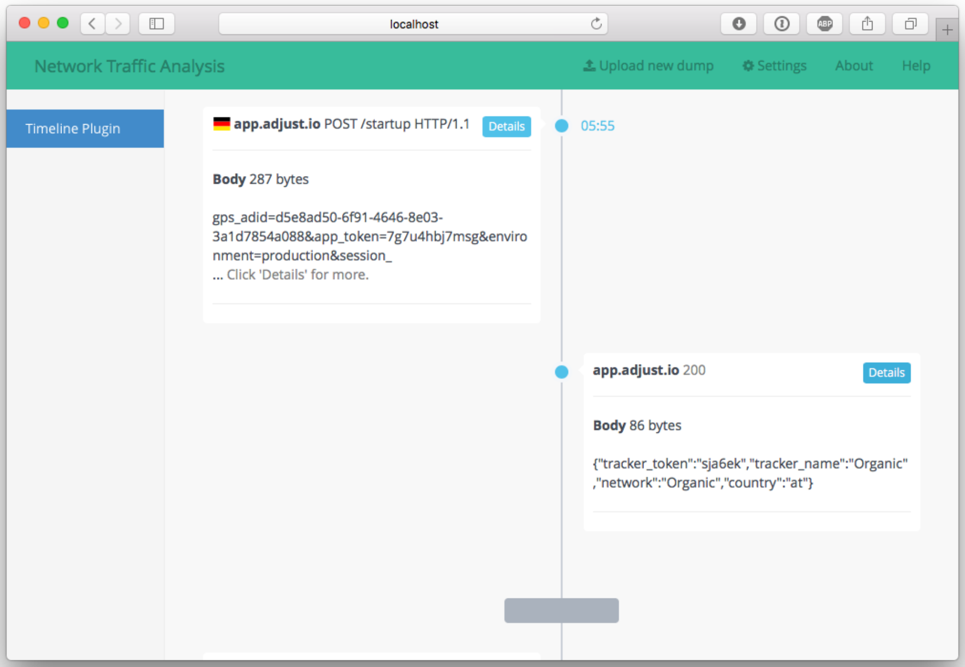
Furthermore, the *LocationLookup* has been extended for AJAX requests in order to be used in the Timeline plugin described in the next Section 5.5. The main purpose of the mentioned plugin was not to detect the origin of the tracking host but during our work we figured out that the information can be very useful for researchers. Thus, we enabled the user to understand in which country a particular network requests terminates. Hence, it is possible to monitor exactly which data has been sent to which country.

5.5 Request-Response Timeline Visualization

During our manual analysis prior writing this thesis, we have recognized that it is important to understand the network communication flow in order to evaluate what an app is doing. Based on this we developed the timeline plugin with this in mind. It visualizes in detail the network requests and responses from the captured traffic and it gives a brief overview about the content of a request. Furthermore, it includes the type and the location of the host and the HTTP parameters. By using this plugin some serious security flaws could be detected. In fact, examples of sensitive data leaks that were monitored by this plugin are presented in the results chapter of this thesis.

The timeline plugin presents the flow of network requests and responses of the analyzed application. Moreover, it uses the aforementioned AJAX extension of the *Network Destination Location* plugin described in the above Section 5.4 for displaying the country where the request has been sent to. This is shown in Figure 5.12.

In addition, the plugin enables the user to see the details of each request. This information includes all parameters and the entire body of the network request. This functionality is shown in Figure 5.13. The screenshot shows the host, all HTTP parameters and the HTTP body at the bottom of the screen.



app.adjust.io POST /startup HTTP/1.1

Body 287 bytes

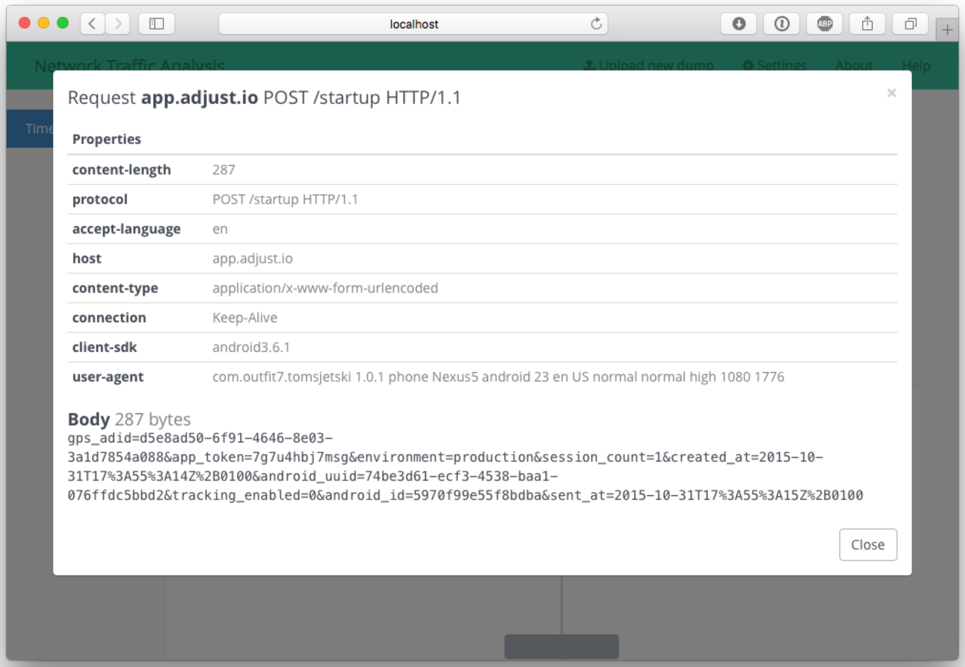
gps_adid=d5e8ad50-6f91-4646-8e03-3a1d7854a088&app_token=7g7u4hbj7msg&environment=production&session_
... Click 'Details' for more.

app.adjust.io 200

Body 86 bytes

{"tracker_token":"sja6ek","tracker_name":"Organic","network":"Organic","country":"at"}

Figure 5.12: Screenshot of the Timeline Visualization Plugin which presents the network flow of the analyzed app.



Request **app.adjust.io** POST /startup HTTP/1.1

Properties	
content-length	287
protocol	POST /startup HTTP/1.1
accept-language	en
host	app.adjust.io
content-type	application/x-www-form-urlencoded
connection	Keep-Alive
client-sdk	android3.6.1
user-agent	com.outfit7.tomsjetski 1.0.1 phone Nexus5 android 23 en US normal normal high 1080 1776

Body 287 bytes

```
gps_adid=d5e8ad50-6f91-4646-8e03-
3a1d7854a088&app_token=7g7u4hbj7msg&environment=production&session_count=1&created_at=2015-10-
31T17%3A55%3A14Z%2B0100&android_uuid=74be3d61-ecf3-4538-baa1-
076ffdc5bbd2&tracking_enabled=0&android_id=5970f99e55f8bdba&sent_at=2015-10-31T17%3A55%3A15Z%2B0100
```

Figure 5.13: Screenshot of the details of a network request within the Timeline Visualization plugin.

6 Evaluation

To increase the reliability of the analysis a representative number of apps was necessary for a significant evaluation. Thus, we have analyzed the top 10 apps from each category of Apple's AppStore and Google's Play Store. In fact, due to the compelling results within the games category, we have selected the top 20 apps from this category for discussing them in this chapter. Furthermore, we describe in detail in this chapter the methodology and experimental setup of this evaluation.

The primary goal of this thesis was to understand the usage of tracking frameworks within numerous of mobile apps. Therefore, we have implemented plugins in order to automatically detect the usage of tracking libraries and to gain insight from the data they are sending to their hosts. One of our findings is the highly usage of unique device identifiers, which enables the providers to identify each particular device among others. Furthermore, the proposed plugins enable the user to understand which and how many tracking frameworks are used by each app, if the data from different apps and providers can be linked together, as also to observe to which destination is the data sent and where are the servers located. Lastly, the plugins enable the user to understand the request-response flow and to gain knowledge about the kind of data that is actually sent. The apps that we have analyzed, were collected from the iOS platform and the Android platform.

The structure of this chapter is as follows. In section 6.1 and 6.2, we describe the methodology and the experimental setup. After that, we present the evaluation goals in Section 6.3.

6.1 Methodology

For the evaluation, 440 distinct apps out of 25 categories on Android and iOS were selected. The apps have been chosen based on their ranking on the AppStore and the Play Store. On iOS as well as on Android some categories have been excluded due to the fact that they do not contain regular apps, such as books and widgets. In fact, on **iOS** the following categories have been excluded from the analysis: *Books, Catalogues, Magazines* and *Newspapers*. On **Android** *Android Wear, Books & Reference, Comics, Google Cast, Libraries & Demos, Live Wallpapers, Personalization* and *Widgets* have been excluded. In addition to the aforementioned categories also the *Family* category has been excluded because it is just a collection of apps from other categories. From each category on both stores 10 apps were collected. However, the categories and number of analyzed apps inside these categories are presented in Table 6.1. In the *Games* category on both platforms 20 apps have been analyzed because of compelling results, which are presented in Chapter 7.

However, some apps used certificate pinning, which is described in Chapter 3. Due to this limitation, which thwarted us from capturing its network traffic, and apps that solely operated offline were regarded as out of scope.

For simplicity and comparability the following 5 Play Store categories were merged into related AppStore categories:

Music & Audio was merged into the AppStore category *Music*.
Photography was merged into the AppStore category *Photo & Video*.
Social was merged into the AppStore category *Social Networking*.
Travel & Local was merged into the AppStore category *Travel*.
Tools was merged into the AppStore category *Utilities*.

Lastly, five interesting aspects, that are related to this thesis, regarding the data within the captured network traffic were identified. The generic core framework, described in Chapter 4, was used for extending it with plugins that analyze and visualize these aspects. This section describes the five evaluation scenarios briefly, which have been applied to the captured network traffic of the beforehand collected apps.

Apple AppStore (iOS)		**Google Play Store (Android)**	
Category	**Number of Apps**	**Category**	**Number of Apps**
Business	10	Business	10
Education	10	Communication	10
Entertainment	10	Education	10
Finance	10	Entertainment	10
Food & Drink	10	Finance	10
Games	20	Games	20
Health & Fitness	10	Health & Fitness	10
Kids	10	Lifestyle	10
Lifestyle	10	Media & Video	10
Medical	10	Medical	10
Music	10	Music & Audio	10
Navigation	10	Photography	10
News	10	Productivity	10
Photo & Video	10	Shopping	10
Productivity	10	Social	10
Reference	10	Sports	10
Shopping	10	Tools	10
Social Networking	10	Transportation	10
Sports	10	Travel & Local	10
Travel	10	Weather	10
Utilities	10		
Weather	10		
Total	**230 Apps**		**210 Apps**

Table 6.1: Categories and number of analysed apps from both app stores

6.2 Experimental Setup

In order to get a representative analysis of mobile apps we decided to conduct this analysis on iOS and Android. We excluded Microsofts Windows Phone operating system from this analysis because of their market share below 1% [41]. In our opinion this number is to less significant in order to conduct an impactful analysis. However, for conducting this analysis on the other two platforms, we used two mobile devices to collect the necessary network traffic. The first is an iPod Touch 5th generation running iOS 9.2 and the second is a Nexus 5 running Android 6.0.1. At the time of writing this both devices ran the newest available version of their operating systems.

We collected the required network traffic by using an SSL Proxy within between the monitored devices and the webservices of the analyzed applications. Further information on what a SSL Proxy is and how it works is described in Section 3.1. For collecting the network traffic we were using Burp Suite, which is described in the next section 6.2.1.

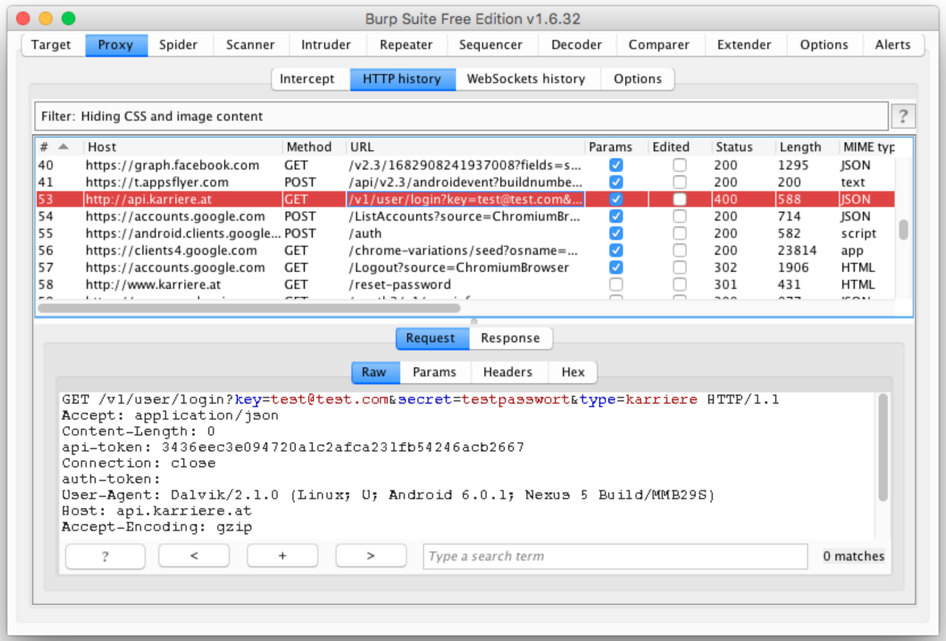
6.2.1 Burp Suite

Burp Suite has been developed by PortSwigger[1] and provides security researchers and analysts with a powerful tool to conduct security testing by monitoring the network traffic. Thus it has been used for collecting the network dumps in order to provide an input for the proposed framework. The only function we used was the so-called *Burp Suite Proxy*, since it was necessary to dump the network traffic for further analysis. [42]

However, Burp Suite offers more components, which are independent from the proposed framework but may be useful for analysts. Hence, they are explained briefly in the following paragraph. [42]

Proxy The proxy enables to intercept and modify HTTP traffic passing through Burp Suite. Moreover, after injecting a self-signed certificate on the test device it is possible to monitor SSL secured connections. In

[1]www.portswigger.net



Filter: Hiding CSS and image content

#	Host	Method	URL	Params	Edited	Status	Length	MIME typ
40	https://graph.facebook.com	GET	/v2.3/1682908241937008?fields=s...	✓	☐	200	1295	JSON
41	https://t.appsflyer.com	POST	/api/v2.3/androidevent?buildnumbe...	✓	☐	200	200	text
53	http://api.karriere.at	GET	/v1/user/login?key=test@test.com&...	✓	☐	400	588	JSON
54	https://accounts.google.com	POST	/ListAccounts?source=ChromiumBr...	✓	☐	200	714	JSON
55	https://android.clients.google...	POST	/auth	✓	☐	200	582	script
56	https://clients4.google.com	GET	/chrome-variations/seed?osname=...	✓	☐	200	23814	app
57	https://accounts.google.com	GET	/Logout?source=ChromiumBrowser	✓	☐	302	1906	HTML
58	http://www.karriere.at	GET	/reset-password	☐	☐	301	431	HTML

```
GET /v1/user/login?key=test@test.com&secret=testpasswort&type=karriere HTTP/1.1
Accept: application/json
Content-Length: 0
api-token: 3436eec3e094720a1c2afca231fb54246acb2667
Connection: close
auth-token:
User-Agent: Dalvik/2.1.0 (Linux; U; Android 6.0.1; Nexus 5 Build/MMB29S)
Host: api.karriere.at
Accept-Encoding: gzip
```

Figure 6.1: Screenshot of active Burp Suite Proxy

addition, particular requests can be sent to other tools in the suite for further analysis.

Spider This tool is based on a crawler that automatically follows links and submits forms. Furthermore, it provides a detailed map of all discovered content. Like in the aforementioned Proxy, it is possible to forward results to other tools.

Scanner This vulnerability scanner is intended for web applications and scans agains all top 10 OWASP[2] vulnerabilities, such as cross-site scripting and SQL injection [43][44]. Moreover, it provides easy-to-use penetration testing, which can be used for quick analysis.

Intruder The intruder provides customized brute-force attacks for exploiting authentication and session handling mechanisms. Arbitrary payloads

[2]www.owasp.org

can be injected and the results of each requests are reported.

Repeater The so-called replay attacks are covered with this tool. It enables the user to resubmit the same request again.

Sequencer The sequencer analyzes the amount of randomness in session identifiers or tokens.

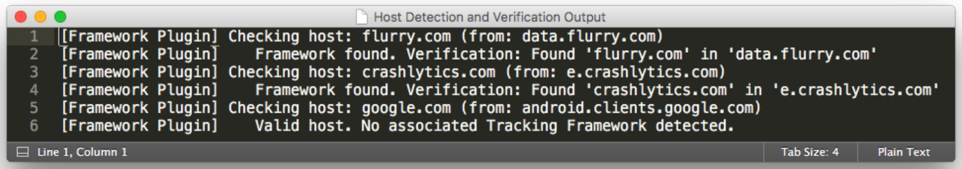
6.3 Evaluation Goals

The proposed framework and the implemented plugins were developed for understanding which kind of data is sent and tracked within the network traffic of mobile apps. Therefore, we have developed the *Analysis of Tracking Frameworks* and the *Detailed Analysis of Tracking Libraries* plugins in order to understand which tracking libraries are used within the collected apps and what kind of data they track. Furthermore, the proposed and implemented *Unique Device Identifier* plugin is intended to give insight on how tracking providers use unique device identifiers for identifying the collected data. The *Location Destination* plugin is used for understanding where the destination servers are located in order to understand the privacy implications of hosted data in other countries. Lastly, the *Timeline* plugin has been implemented in order to enable the user to conduct a detailed analysis of the traffic and its request-response timeline. The aforementioned plugins are described in Chapter 5 and their results are presented in Chapter 7.

The main purpose of this thesis was to evaluate and understand the usage of tracking libraries within mobile apps on the iOS and the Android platform. Furthermore, another primary goal was to identify unique device identifiers and their usage across multiple apps in the network traffic of the collected apps. The next section presents the evaluation goals for the usage of tracking frameworks within the mobile apps. After that, we describe the detection of unique device identifiers within the collected apps and its limitations.

6.3.1 Detection of Tracking Frameworks

The main purpose of this thesis is to identify the usage of tracking frameworks on mobile platforms. Therefore, we analyzed the existing frameworks by checking which connections are opened by apps on iOS and on Android. We achieved this by creating a rule matching to our framework. The proposed framework and its plugins can detect known tracking and advertisement libraries. *Known-Frameworks* are defined as frameworks that are implemented within the frameworks plugin and are identified by at least one host. We have obtained the frameworks and the known hosts by using a manual analysis of the first 40 network dumps from the games category. In order to detect the tracking providers we have introduced a hosts file, which is shown in Figure 6.2, that generates the output and logs if a framework has been matched with the given host. Thus, we have controlled our matching algorithm by manually verifying the detected frameworks for the first 40 apps. Furthermore, also the non-matching hosts are tracked in order to identify and evaluate potential tracking hosts for the future. Thus, the framework and its plugins detect all ascertained tracking and advertisement providers identified by their manually collected hosts.



```
[Framework Plugin] Checking host: flurry.com (from: data.flurry.com)
[Framework Plugin]    Framework found. Verification: Found 'flurry.com' in 'data.flurry.com'
[Framework Plugin] Checking host: crashlytics.com (from: e.crashlytics.com)
[Framework Plugin]    Framework found. Verification: Found 'crashlytics.com' in 'e.crashlytics.com'
[Framework Plugin] Checking host: google.com (from: android.clients.google.com)
[Framework Plugin]    Valid host. No associated Tracking Framework detected.
```

Figure 6.2: Screenshot of the Host Detection and Verification Output

Considering, that the algorithm matches the detected host with the deposited host in the plugin implementation, we conclude that the algorithm is reliable. Due to the fact, that we detect tracking and advertisement frameworks by matching the host of the network request, there are no false positives in our results.

However, our proposed framework can only detect known frameworks which were implemented within the plugins. In-house tracking or proprietary tracking libraries are not covered in the implementation and thus they are not part of this thesis. Another limitation is that the matching is bound to a particular host. If the provider changes the host, then the corresponding request will not be detected as a tracking request anymore.

6.3.2 UDID Detection

Another interesting aspect is the usage of unique device identifiers, the so-called *UDIDs*, within mobile apps. We have encountered this during our manual analysis prior writing this thesis. We were able to find a possible correlation between so called *Unique Device Identifiers* across multiple apps. Thus, we have implemented a plugin that analyzes this behavior automatically and detects if the providers are able to use the same identifier on different mobile apps or not.

The plugin uses a *Regular Expression* for detecting unique device identifiers. The identifiers are based on 32 characters using a particular pattern. The pattern approach has been chosen because of the standardized representation of these identifiers. Since we are only analyzing detected identifiers that are matching the aforementioned pattern, there are no false-positives in our results.

However, some limitations of this methodology is the ability of the providers to structure their identifier on a different way or obfuscate the obtained device identifier before sending it to their servers. Moreover, identifiers that are encoded on a different way cannot be observed by using this approach.

7 Results

In this chapter we present the results of our analysis of the apps mentioned in Section 6.1. The results are structured according to the evaluation scenarios defined in Section 6.3. All results have been obtained by using the automated analysis of the plugins described in Chapter 5. We have combined the independent results of different plugins in order to create connected results.

This chapter is structured as follows: In Section *Framework Usage in Analyzed Apps* we describe the combined results of the plugins *Analysis of Tracking Frameworks* and *Detailed Analysis of Tracking Libraries*. Both plugins are described in Section 5.1 and 5.2. After that, we present in Section 7.2 the *Usage of Unique Device Identifiers* and in Section 7.3 the findings of the *Network Destination Location* plugin. As mentioned in Chapter 5, we have elaborated all the results we observed in the *Timeline* plugin. In Section 7.4, we demonstrate and discuss the most interesting and representative findings of the *Timeline* plugin. The last section of this chapter describes the usage of *Certificate Pinning* that we were able to observe during our collection phase.

As briefly described above, this chapter presents the results and findings from the implemented plugins. We decided to include the discussion about the findings in the corresponding sections after presenting the results objectively.

7.1 Framework Usage in Analyzed Apps

This section covers the results of the *Analysis of Tracking Frameworks* plugin and the *Detailed Analysis of Tracking Libraries* plugin. Furthermore, we have analyzed the apps of every AppStore/PlayStore category independently and we present our findings in this section. In total we have analyzed 440 apps from 25 distinct categories. Therefore, 230 of them were on iOS and 210 on Android. This is due to the fact that Apple's AppStore offered more categories than Google's PlayStore. For our purposes, we have dumped the top ten apps from each category except the games category. Due to our significant findings we have recorded twice as much apps from the games category, than from the other categories.

The first part of our results is presented in Table 7.1. The first column illustrates the categories that were analyzed. The second and third column states the number of apps, which were dumped inside each category. We have captured the traffic of the top ten or top twenty apps of each category.

The other columns show the number of tracking frameworks found inside the apps of each category. More particularly, they illustrate how many tracking libraries were found in total in each category and how many of them were distinct. Hence, we distinguish between the total number of frameworks found and the number of unique frameworks per category. This enabled us to compare the categories with each other. Furthermore, the table shows the findings per platform. Thus, we were able to compare the iOS and Android platform with each other with respect to how many different tracking frameworks were found on each platform. For example, in the first category *Business* 10 iOS and 10 Android apps have been analyzed. Within these 10 iOS apps our plugins have detected 15 tracking frameworks in total, whereby 9 of them were distinct. This means inside 10 apps we have detected 9 different tracking libraries. However, on Android we have detected 19 in total and 8 of them were distinct. Lastly, we present in the last row at the bottom of the table, the number of total tracking frameworks found inside all categories.

In addition, our analysis has also covered the occurrence of the tracking frameworks inside the apps. Therefore, we have selected all the apps from a particular category and from one platform and we have used the two aforementioned plugins for our purposes. Therefore, we were able to monitor in how many apps from our input we could detect at least one tracking library inside the network traffic. This number is particularly interesting because it has enabled us to understand how prevalent such libraries are.

	# of Apps		**Tracking Frameworks found**			
Category	**iOS**	**Android**	**iOS**	**Distinct**	**Android**	**Distinct**
Business	10	10	15	**9**	19	**8**
Education	10	10	20	**8**	14	**4**
Entertainment	10	10	38	**21**	31	**17**
Finance	10	10	14	**9**	12	**5**
Food & Drink	10	-	14	**8**	-	**-**
Games	20	20	124	**38**	98	**36**
Health & Fitness	10	10	18	**9**	25	**10**
Kids	10	-	21	**15**	-	**-**
Lifestyle	10	10	18	**8**	16	**9**
Medical	10	10	10	**6**	20	**10**
Music	10	10	34	**16**	24	**11**
Navigation	10	-	14	**11**	-	**-**
News	10	-	13	**5**	-	**-**
Photo & Video	10	10	20	**11**	27	**10**
Productivity	10	10	15	**9**	23	**12**
Reference	10	-	31	**15**	-	**-**
Shopping	10	10	24	**12**	21	**9**
Social Networking	10	10	20	**8**	31	**9**
Sports	10	10	14	**9**	13	**6**
Travel	10	10	7	**4**	17	**8**
Utilities	10	10	25	**15**	24	**11**
Weather	10	10	27	**12**	19	**7**
Communication	-	10	-	**-**	14	**8**
Media & Video	-	10	-	**-**	12	**6**
Transportation	-	10	-	**-**	6	**5**
Total	**230**	**210**	**536**		**466**	

Table 7.1: Total and Distinct Tracking Frameworks found inside Apps' Network Traffic

The result of this analysis is shown in Table 7.2. Like in the previous table, the first columns show the analyzed categories and the number of analyzed apps per platform. The fourth column shows the number of iOS apps in which at least one tracking framework was detected. The fifth column shows the percentage of iOS apps, which contained at least one tracking library. The next two columns show the same figures but for the Android platform. The last column shows in percentage the difference of detected tracking frameworks between the two platforms iOS and Android. Thus, we were able to compare both platforms with each other for every category. Furthermore, at the bottom of the table in the last row we present the average number of apps per platform containing at least one tracking library and the average difference of detected tracking frameworks between iOS and Android.

Let us consider the following example: In the second category *Education* we were able to detect 8 apps that used at least one tracking framework on iOS. These form the 80% of all the analyzed apps inside this category. On the Android platforms the plugins detected 9 frameworks, which represent 90% of all apps. This is a difference of 10% between iOS and Android.

Discussion. The results presented in Table 7.1 and Table 7.2 show the number of tracking frameworks that we were able to detect on iOS and on Android. In fact, we were only able to detect the ones, which have been registered within the plugins. However, this limitation is described in the implementation of the plugin in Section 5.2.

We were able to detect in total 124 tracking frameworks within the *Games* category on the iOS platform, whereby 38 out of these 124 frameworks were distinct. We show, that our plugins have detected 38 distinct tracking frameworks within only 20 iOS apps. This number is almost twice the number of the total apps. Statistically, every app is using approximately two tracking libraries. Moreover, Table 7.2 shows the number of apps, which are using at least one tracking framework. Interestingly, 10 out of 10 apps in this category are using at least one tracking library, which is 100% of all apps on the iOS platform in this category. Considering the aforementioned number of distinct frameworks that we have detected in those 20 apps, we conclude that some apps are using multiple frameworks for the same or for different purposes. Furthermore, we have recognized that many apps

	# of Apps		Containing FWs				
Category	**iOS**	**Andr.**	**iOS**		**Android**		**Diff**
Business	10	10	8	**80%**	8	**80%**	0%
Education	10	10	8	**80%**	9	**90%**	-10%
Entertainment	10	10	9	**90%**	7	**70%**	20%
Finance	10	10	7	**70%**	7	**70%**	0%
Food & Drink	10	-	7	**70%**	-	**-**	-
Games	20	20	20	**100%**	19	**95%**	5%
Health & Fitness	10	10	7	**70%**	8	**80**	-10%
Kids	10	-	9	**90%**	-	**-**	-
Lifestyle	10	10	7	**70%**	7	**70%**	0%
Medical	10	10	5	**50%**	8	**80%**	-30%
Music	10	10	10	**100%**	10	**100%**	0%
Navigation	10	-	6	**60%**	-	**-**	-
News	10	-	8	**80%**	-	**-**	-
Photo & Video	10	10	9	**90%**	9	**90%**	0%
Productivity	10	10	9	**90%**	9	**90%**	0%
Reference	10	-	10	**100%**	-	**-**	-
Shopping	10	10	9	**90%**	9	**90%**	0%
Social Networking	10	10	8	**80%**	10	**100%**	-20%
Sports	10	10	8	**80%**	8	**80%**	0%
Travel	10	10	5	**50%**	8	**80%**	-30%
Utilities	10	10	5	**50%**	9	**90%**	-40%
Weather	10	10	8	**80%**	10	**100%**	-20%
Communication	-	10	-	**-**	9	**90%**	-
Media & Video	-	10	-	**-**	9	**90%**	-
Transportation	-	10	-	**-**	3	**30%**	-
Total	**230**	**210**		78%		83%	**-8%**

Table 7.2: Number of Apps containing at least one Tracking Framework

are using multiple advertisement libraries in order to display the most profitable advertisement from them. Hence, the data from one app can be sent to multiple providers. The numbers also show that every analyzed app in the games category is tracked. Initially, we have started analyzing 10 apps from that section but then we detected that every app is using at least one tracking framework, so we have extended our analysis for the top 20 apps of this section. Impressively, on iOS there was not even one app in this category that did not use a tracking or an advertisement library.

On Android we were able to confirm the numbers from the iOS platform. There we were able to identify 98 frameworks in total in the games category, whereby only 36 of them were distinct. Furthermore, on Android our plugins have detected that 19 out of 20 apps are using at least one tracking framework. This is 95% of all Android apps in this category.

The games category was an extraordinary example because even if we have used the local app stores, most of the apps in this category were developed by professional game development studios. This category is certainly one with the most revenue in the stores. Hence, this category is of major interest for the companies. Thus, the tracking and analytics might be very important in order to increase the conversion rate and the revenue per user.

Interestingly, we were able to observe that there is not a single category without the usage of a tracking or an advertisement framework. The results clearly show that tracking is very common in mobile app development. Table 7.1 illustrates that although there are differences in the categories, there is no category without tracking. Obviously, entertainment, games, music and reference categories are of great interest for the vendors. In those categories we were able to find more than 30 tracking libraries, whereby the peak was in the games category with 124 frameworks on iOS and 98 on Android.

The results in Table 7.2 also show that there is a very high amount of apps, with at least one tracking framework. On iOS we were able to observe that 100% of the analyzed apps in the categories of games, music and reference have used the aforementioned type of frameworks. In comparison, to Android the categories music, social networking and weather have a framework usage per app of 100%. Considering the categories with a usage of more than 70%, on iOS only four categories are below of this level and on Android only one single category. In the transportation category on the Android platform, 3 out of 10 apps have used at least one tracking framework. Hence, this is only 30% of the total apps within this category. This is an outliner among our data and can be described because of the apps in this category are selected from the local Austrian app store. Apps that are related to transportation are only usable in a local or nationwide scope. Thus, there is no major interest in this category. The same rule applies to the same category on the iOS platform; it is the so-called navigation category. There we were able to observe a usage of 60%, which is more than on Android but still below the average on the iOS platform.

Basically, the numbers we observed on iOS are similar to the ones on Android. The only category with a significant difference of 40% is the utilities category. On iOS we were able to detect in the utilities category, that 50% of the apps are using at least one tracking framework, whereby on Android 90% of the apps utilized at least one.

Our results show that the difference between iOS and Android is only 5% on average. Table 7.2 illustrates that on average 78% of the apps across all categories are using at least one tracking library, whereby on the Android platform 83% are using at least one framework. The numbers show that the usage is almost the same on both platforms. Hence, we have observed that there is a focus on particular categories but not on platforms. Considering both platforms, our analysis has also evidenced that there are no significant differences in the categories, except of three statistical outliners.

Considering these results, certainly there is a massive interest in collecting information about the devices and its users'. The collected data from just a single app is statistically unappealing for the tracking providers. Thus, it is important for tracking and advertisement providers to collect the data from multiple apps. According to the results shown in Table 7.1, there are much more detected frameworks in each category than distinct ones. This implies that one or more tracking libraries are used multiple times across different apps in the same category. Hence, the providers gather data from the same user by using different apps. However, the data is uncritical as long as it cannot be connected across the multiple apps. In fact, this forms a critical question in our thesis. For that reason, we have developed the *Usage* *of Unique Device Identifiers* described in Section 5.3. The results from this plugin and the answer to our question is described in the next Section 7.2.

7.2 Usage of Unique Device Identifiers

During our analysis we have realized how important unique device identifiers are for advertisement providers. These identifiers are also known as UDIDs. In fact, these identifiers transform arbitrary data to something meaningful. The results of our *Usage of Unique Device Identifiers* plugin are presented in this section and interpreted in the discussion paragraph at the bottom of this section. The results include the analysis of 440 apps in total, whereby 230 were analyzed on iOS and 210 on Android. The differences among the number of apps result from different categories in the devices' app stores. As described in Section 6.1, we have merged categories together across the different stores wherever it was possible.

The results of this plugin serves two purposes. The first is to detect how many unique device identifiers have been used in the iOS and Android apps in each category. Secondly, considering that such identifiers enable to uniquely link data together that has been sent to third-party servers, it is particularly interesting to observe if the same identifier has been used in multiple apps. This might enable advertisers to link data together from different apps, within multiple categories.

Table 7.3 shows the analyzed categories and the number of monitored apps inside each category. Moreover, the table shows how many apps from each platform have been analyzed in every category. The last two columns show how many unique identifiers of the analyzed apps have been found within the network traffic. We distinguish between detected identifiers on iOS and on Android in order to be able to compare the results between the two different platforms. For example, in the third category *Entertainment* we have analyzed 10 apps from iOS and 10 from Android. Within the network traffic of the iOS apps in this category we have detected 28 unique identifiers. On Android we have detected 20 unique identifiers.

	# of Apps		UDIDs Found	
Category	**iOS**	**Android**	**iOS**	**Android**
Business	10	10	**16**	**19**
Education	10	10	**39**	**9**
Entertainment	10	10	**28**	**20**
Finance	10	10	**11**	**14**
Food & Drink	10	-	**18**	**-**
Games	20	20	**98**	**45**
Health & Fitness	10	10	**20**	**20**
Kids	10	-	**21**	**-**
Lifestyle	10	10	**29**	**14**
Medical	10	10	**19**	**20**
Music	10	10	**35**	**17**
Navigation	10	-	**28**	**-**
News	10	-	**14**	**-**
Photo & Video	10	10	**52**	**11**
Productivity	10	10	**20**	**13**
Reference	10	-	**24**	**-**
Shopping	10	10	**30**	**24**
Social Networking	10	10	**19**	**25**
Sports	10	10	**24**	**8**
Travel	10	10	**39**	**19**
Utilities	10	10	**24**	**4**
Weather	10	10	**22**	**5**
Communication	-	10	**-**	**13**
Media & Video	-	10	**-**	**15**
Transportation	-	10	**-**	**1**
Total	**230**	**210**		

Table 7.3: Unique Device Identifiers found in each Category

After having detected how many unique device identifiers have been used, we have used the *Usage of Unique Device Identifiers* plugin for analyzing and detecting how often the same identifier was used within multiple apps. In fact, we have tracked if the same unique identifier occurred multiple times in one or more different apps, by analyzing the network traffic of each app. These results are illustrated in Table 7.4. Like in the previous table, the first column shows the different categories from the app stores. The third and fourth column describes how many iOS and Android apps have been analyzed within each category. The other columns show the number of apps, which have used the same UDID. For example, we analyzed 10 iOS and 10 Android apps from the *Finance* category by using our plugin. As illustrated in Table 7.3, we have detected 11 UDIDs on iOS and 14 on Android. One out of this 11 detected identifiers on iOS have been used in 6 apps. Hence, 60% of the apps analyzed in this category tracked data with the same identifier. This implies that this data can be linked together on the servers. On Android at least one shared identifier was used in 4 different apps.

In case of multiple matches, we have also included more than one identifier in our results. If we were able to find more identifiers that we have used in multiple apps, we reported this in the next column. In fact, in some categories we were able to detect 4 different identifiers that were used in at least 2 different apps. Due to space limitations we only show the first two matches in Table 7.4. The full table, including all results, is available in the Appendix.

Discussion. Our results in Table 7.3 show that there is a massive use of unique device identifiers inside the network traffic. The numbers we found correlate with the findings described in Section 7.1. In the games category we were able to detect the most usage of UDIDs on iOS. Although we have analyzed 20 apps in this category, the results show relatively high numbers with 98 detected device identifiers within 20 apps. However, on Android we have observed only 45 identifiers. In many categories we have detected significantly less UDIDs on Android than on iOS. For example, in the Photo & Video category we have monitored 52 identifiers on iOS and only 11 on Android. Furthermore, in the Weather category we have detected 5 UDIDs on Android, but 22 on iOS. Considering the results illustrated in Table 7.3, where the usage of frameworks was similar on iOS and on Android, we

	# of Apps		**Same UDID used by # of Apps**							
Category	**iOS**	**Andr.**	**iOS**				**Android**			
Business	10	10	5	**50%**	-	**-**	4	**40%**	2	**20%**
Education	10	10	7	**70%**	-	**-**	4	**40%**	-	**-**
Entertainment	10	10	7	**70%**	-	**-**	6	**60%**	-	**-**
Finance	10	10	6	**60%**	-	**-**	4	**40%**	-	**-**
Food & Drink	10	-	5	**50%**	-	**-**	-	**-**	-	**-**
Games	20	20	17	**85%**	3	**15%**	18	**90%**	2	**10%**
Health & Fitn.	10	10	6	**60%**	2	**20%**	7	**70%**	-	**-**
Kids	10	-	8	**80%**	3	**30%**	-	**-**	-	**-**
Lifestyle	10	10	5	**50%**	-	**-**	6	**60%**	-	**-**
Medical	10	10	3	**30%**	-	**-**	4	**40%**	-	**-**
Music	10	10	7	**70%**	-	**-**	5	**50%**	-	**-**
Navigation	10	-	4	**40%**	-	**-**	-	**-**	-	**-**
News	10	-	5	**50%**	-	**-**	-	**-**	-	**-**
Photo & Video	10	10	7	**70%**	-	**-**	6	**60%**	-	**-**
Productivity	10	10	5	**50%**	3	**30%**	5	**50%**	-	**-**
Reference	10	-	5	**50%**	-	**-**	-	**-**	-	**-**
Shopping	10	10	8	**80%**	-	**-**	8	**80%**	-	**-**
Social Netw.	10	10	8	**80%**	-	**-**	9	**90%**	-	**-**
Sports	10	10	5	**50%**	-	**-**	2	**20%**	-	**-**
Travel	10	10	4	**40%**	-	**-**	5	**50%**	-	**-**
Utilities	10	10	5	**50%**	-	**-**	5	**50%**	-	**-**
Weather	10	10	6	**60%**	2	**20%**	3	**30%**	-	**-**
Communication	-	10	-	**-**	-	**-**	4	**40%**	-	**-**
Media & Video	-	10	-	**-**	-	**-**	2	**20%**	-	**-**
Transportation	-	10	-	**-**	-	**-**	1	**10%**	-	**-**
Total	**230**	**210**		**59%**		**23%**		**50%**		**15%**

Table 7.4: Same UDID used by more than one App

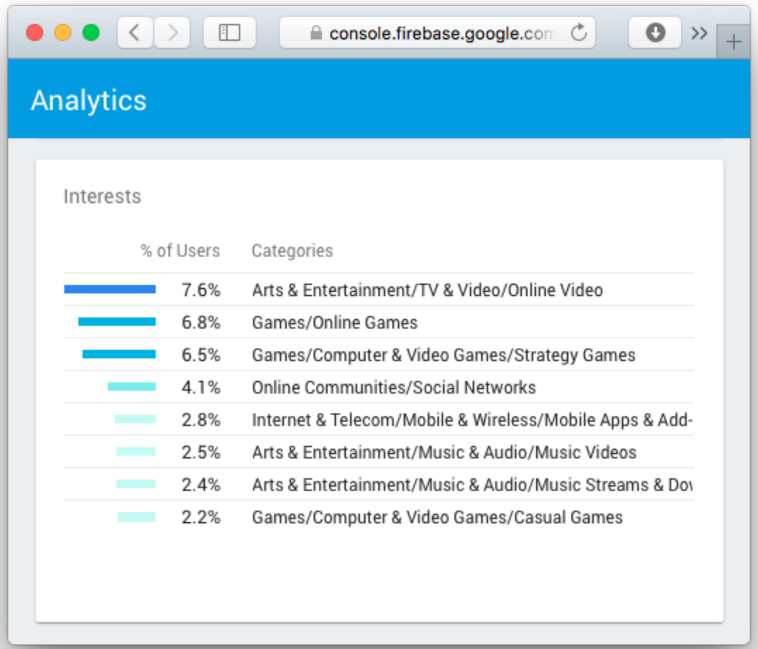
assume that the tracking and advertisement providers might have an easier access to unique identifiers on Android than on iOS. This shows that it is easier to derive a unique device identifier on Android than on iOS. As a result, the providers tend to use the same identifier. In fact, this might be critical because it enables the providers to link the gathered data from multiple apps to one particular entity. In other words, the more data they collect, the more accurate the persona becomes.

Table 7.4 shows the number of apps which have used the exactly same identifier within their network request. This outcome is very significant, due to the fact that it enables tracking providers to connect data from different apps to one particular entity. Hence, the accuracy of the described entity increases with the collected data. If it is possible to derive unique identifiably numbers, such as the social insurance number or names from this entity, it is also then possible to identify a particular person. However, the providers use these entities to derive personas which describe a particular group of users with the same character and background. For instance, Google Firebase and Doubleclick provide to the app developers information about the location, the gender, the age and the interests of their users. The interests are accurate and narrow down special activities of the users. An example of tracked interests that Firebase provides is shown in Figure 7.1.

Moreover, we have identified that in most categories more than 50% of all apps within this category, are using the exactly same device identifier. Considering that most people do not own multiple mobile phones, this device identifier can also be seen as a personal identifier. Thus, it identifies the individual person and not just a device. Furthermore, due to device backups in the cloud we still have the same identifier even when we buy a new device. Furthermore, like in the previous results, we have clearly identified a peak in the games category, where we were able to detect 4 distinct UDIDs that have been used by multiple apps on iOS and 3 distinct identifiers on the Android platform. In addition, the numbers among iOS and Android correlate, so there are not significant outliners between the platforms. In the education, entertainment, games, kids, music, photo & video, shopping and social networking categories on iOS we were able to identify significant numbers of 70% and more. This means that 70% of all apps in this category are sharing at least one unique identifier for identifying this device. On contrary, on the Android platform, we have detected the categories games, shopping and social networking with a usage of 70% and more. The average percentage on iOS was 59% and on Android 50%. Thus, both platforms are affected in almost the same way.

In the games category our combined results illustrate that on iOS 85% and on Android 90% of all apps in this category are using the same unique identifier. Hence, it is possible to connect the data from almost all analyzed games with each other. Moreover, in the games category we were able to observe 4 different identifiers that were used within at least 2 distinct apps. For instance, on iOS 17 apps have used the same identifier. We have also detected another identifier, which was exactly the same within 3 distinct apps. Moreover, we detected another two identifiers that were also used in two different apps. Thus, we were able to prove that not only the advertisement identifier is used from multiple providers, but there are also other device identifiers which are unique and are also used by the providers. Considering, that these identifiers have been used within different apps, it enables the companies which are providing these tracking and advertisement libraries, to connect data from multiple apps.

However, the contextual information about a particular user becomes more accurate if the collected data came from different categories. Our results demonstrate that these tracking frameworks are collecting data from all categories and are able to link these data together. For instance, the behavior of a user playing a particular game is not very critical. But whenever there is data collected from different contexts, the persona becomes more transparent. Thus, when the aforementioned behavioral data is linked with shopping preferences from a shopping app and furthermore, if it is also connected to the person's spending attitude or travel preferences from a travel app, the person becomes less abstract. All this information can draw a pretty accurate persona. Basically, the only missing part is an identifiable attribute in order to derive a real person. In fact, many apps ask for the user's email address, name or physical address. In that way, they are able to uniquely identify a person and based on the collected data, they can extract all the information about the person's job, salary, friends, preferences and much more.



Interests

	% of Users	Categories
▬	7.6%	Arts & Entertainment/TV & Video/Online Video
▬	6.8%	Games/Online Games
▬	6.5%	Games/Computer & Video Games/Strategy Games
▬	4.1%	Online Communities/Social Networks
▬	2.8%	Internet & Telecom/Mobile & Wireless/Mobile Apps & Add-
▬	2.5%	Arts & Entertainment/Music & Audio/Music Videos
▬	2.4%	Arts & Entertainment/Music & Audio/Music Streams & Do
▬	2.2%	Games/Computer & Video Games/Casual Games

Figure 7.1: Screenshot of the users tracked interests in Google Firebase Analytics

7.3 Network Requests' Destinations

With different privacy laws in place, it became important to understand where the data is transferred to. Since most apps use third party libraries for tracking, analytics or as advertisement providers, the data is not kept at the device anymore. We found out that most tracking libraries are developed by companies in the US and therefore, the tracked data is stored there. We implemented the *Network Destination Location* plugin in order to understand and monitor where the data is sent to. As explained in Section 5.4 the plugin analyzes every outgoing request from the client and tracks the host's destination via IP lookup. Thus, we were able to understand how many requests from each app were sent to particular countries.

The results of this plugin are shown in Table 7.5. The table illustrates only the two countries with the most requests. For all the other countries and for any further details the full table is attached in the Appendix. More specifically, the table shows the analyzed categories in the first column. The second and third column states the number of apps that we have analyzed. We have used the aforementioned *Network Location Destination* plugin for collecting the results shown in the other columns. The Table in Figure 7.5 uses two separate columns per platform. The first column of each platform is for the *United States* and the second for *Austria*. Therefore, we used the short codes of each country for identifying them. Hence, *US* stands for *United States* and *AT* for *Austria*. Both countries are shown in this table, since they were first and second in our results. In total, all apps analyzed, have sent the most requests to the US and the second most ones to Austria. Furthermore, we illustrate the total number of apps, which have sent most of their requests to this particular country in the last row of the table.

For example, in the Business category we have analyzed 10 iOS and 10 Android apps. The next column indicates that 6 out of those 10 iOS apps have sent the vast majority of their requests to hosts located in the *United States*. This is indicated by the short code *US* in the table header. Furthermore, 2 out of those 10 apps have sent most of their requests to Austrian servers. The 2 other apps, which are left out of 10, have used one server in Germany and one in France. However, this is not shown in Table 7.5 because of limited space, but it is shown in the entire table in the Appendix. Moreover, 7 out of 10 Android apps have also used servers located in the US as their primary destination for their network requests. One app has used a host that was located in Austria, as shown in the Android columns.

Discussion The results presented in Table 7.5 clearly show that in total the most requests end in the United States. Thus, most data is transferred to the US which might have privacy implications because of different privacy laws. Furthermore, we have detected that the second most sent requests are targeting to Austrian servers. We have found out that this is because of the local app stores. All mobile app stores use local rankings in order to provide the most interesting apps to their users. Thus, our analysis covered the top apps from the Austrian stores. Hence, many local apps request resources from servers inside Austria, such as the public transportation apps. Furthermore, our hypothesis was verified via the news category on

	# of Apps		Major Hosts			
			iOS		Adroid	
Category	**iOS**	**Andr.**	**US**	**AT**	**US**	**AT**
Business	10	10	6	2	7	1
Education	10	10	3	1	8	1
Entertainment	10	10	8	-	6	-
Finance	10	10	5	2	3	4
Food & Drink	10	-	6	2	-	-
Games	20	20	18	-	20	-
Health & Fitn.	10	10	5	-	8	-
Kids	10	-	8	-	-	-
Lifestyle	10	10	5	2	5	2
Medical	10	10	4	-	9	-
Music	10	10	5	-	7	1
Navigation	10	-	4	4	-	-
News	10	-	1	7	-	-
Photo & Video	10	10	8	-	9	-
Productivity	10	10	5	-	7	-
Reference	10	-	10	-	-	-
Shopping	10	10	7	1	6	1
Social Netw.	10	10	7	-	7	-
Sports	10	10	4	2	2	3
Travel	10	10	3	2	5	3
Utilities	10	10	5	2	5	1
Weather	10	10	4	4	7	2
Communication	-	10	-	-	6	1
Media & Video	-	10	-	-	5	1
Transportation	-	10	-	-	3	4
Total	**230**	**210**	**131**	**31**	**135**	**25**

Table 7.5: Number of Apps that have sent the most requests to Hosts in the mentioned Countries

iOS, where 70% of the analyzed apps have transmitted the most data to servers in Austria.

Interestingly, on Android all apps in the games section have used servers located in the US as their primary destination, as also almost all games on iOS. Moreover, 100% of all apps in the reference category have used servers in the US as their primary source, which is because of the translation apps in this category. Since the primary translation language is English, the apps in the reference category are using servers in the US.

Furthermore, in the productivity category on Android we were able to detect 3 apps which have used Chinese servers as their primary destination, which is not something uncommon. However, one particular app has served a very specific purpose that should normally not require any Internet connection. However, the app has sent massive data to servers located in China. Thus, we have investigated and analyzed the app further by using the *Timeline* plugin, which is described in the next Section.

7.4 Results from the Timeline Plugin

Beside the aforementioned plugins, we have also developed another plugin that visualizes the captured traffic in an intuitive way. The plugin uses a timeline structure for displaying each request and the corresponding response from the server. This enables the user to analyze the data flow in detail. We used this plugin to analyze apps with suspicious behavior, such as apps sending massive data to foreign hosts or using multiple tracking libraries. Thus, we developed the *Timeline* plugin and used it for manually analyzing these apps.

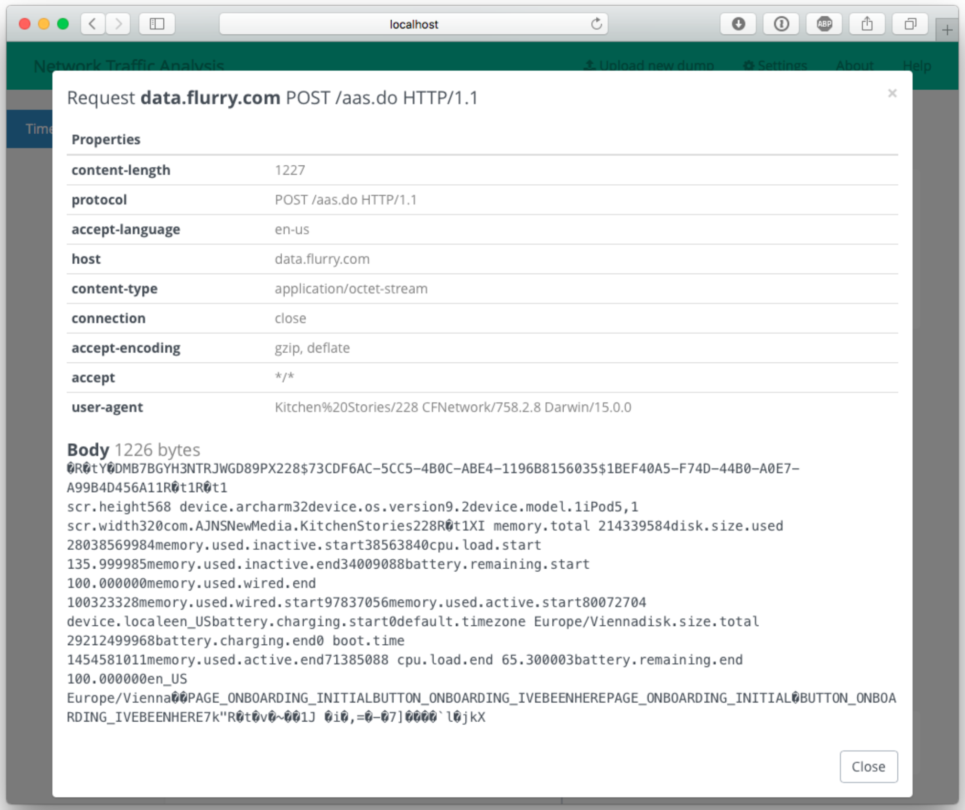
Basically, the *Timeline* plugin was the first plugin we have developed. It was our starting point in order to gain insight in what actually happens in the network traffic of the apps. By using this plugin we figured out that the hosts' location might be interesting. Thus, we developed the *Network Destination Location* plugin described in Section 5.4 and the results are presented in the previous Section 7.3. After that, we realized that the majority of the traffic within some apps is sent to tracking or to advertisement hosts. Therefore, we implemented the Tracking Frameworks plugins described in Section 5.1 and 5.2. Furthermore, we were able to detect the so-called unique device identifiers inside the body of the network requests and implemented the UDID plugin described in Section 5.3. In fact, the results from the *Timeline* plugin were the starting point for the other analysis and results.

7.4.1 Tracked Data

One of our findings based on the *Timeline* plugin, is that information about the devices and their users' is sent to tracking servers. One example for such a tracking framework is *Flurry*[1]. This analytics and advertisement library collects extensive data about the device. It tracks device information such as memory usage, cpu usage, battery status and every button that was pressed by the user inside the app. The results from the timeline plugin for a specific request to Flurry is shown in Figure 7.2. In the beginning of every request it sends the user ID that it has assigned to this particular device. Then, it sends information about which app is tracked and the aforementioned information about the device and the user. Furthermore, we found out that it also sends the user's time zone.

Discussion. We analyzed selected apps individually by using the *Timeline* plugin. In the *Food & Drink* category we have detected one app that uses the tracking framework *Flurry* extensively. The framework tracks the device information, such as the screen height and width, the devices architecture, the processor, the exact device model, the memory usage before a view is presented and the memory usage afterwards. Furthermore, our analysis has shown that it tracks the battery level for every state of the user interface transition and for the CPU usage as well. In addition, we observed that this particular app tracks every button that has been clicked by the user. Hence, every path the a user takes through the app is tracked for unknown purposes on their servers. The app is intended for presenting cooking recipes to the user. However, the company which has developed this app, monitors every clicked recipe and even also the touches inside the recipe.

[1]http://www.flurry.com



Request **data.flurry.com** POST /aas.do HTTP/1.1

Properties	
content-length	1227
protocol	POST /aas.do HTTP/1.1
accept-language	en-us
host	data.flurry.com
content-type	application/octet-stream
connection	close
accept-encoding	gzip, deflate
accept	*/*
user-agent	Kitchen%20Stories/228 CFNetwork/758.2.8 Darwin/15.0.0

Body 1226 bytes

```
�R�tY�DMB7BGYH3NTRJWGD89PX228$73CDF6AC-5CC5-4B0C-ABE4-1196B8156035$1BEF40A5-F74D-44B0-A0E7-
A99B4D456A11R�t1R�t1
scr.height568 device.archarm32device.os.version9.2device.model.1iPod5,1
scr.width320com.AJNSNewMedia.KitchenStories228R�t1XI memory.total 214339584disk.size.used
28038569984memory.used.inactive.start38563840cpu.load.start
135.999985memory.used.inactive.end34009088battery.remaining.start
100.000000memory.used.wired.end
100323328memory.used.wired.start97837056memory.used.active.start80072704
device.localeen_USbattery.charging.start0default.timezone Europe/Viennadisk.size.total
29212499968battery.charging.end0 boot.time
1454581011memory.used.active.end71385088 cpu.load.end 65.300003battery.remaining.end
100.000000en_US
Europe/Vienna��PAGE_ONBOARDING_INITIALBUTTON_ONBOARDING_IVEBEENHEREPAGE_ONBOARDING_INITIAL�BUTTON_ONBOA
RDING_IVEBEENHERE7k"R�t�v�~��1J �i�,=�-�7]����`l�jkX
```

Figure 7.2: Screenshot of the output of the Timeline plugin showing a request sent to Flurry

When analyzing such apps we were also able to detect every information that the client sends to the backend servers. Thus, in some apps we observed the transmission of the users' password in plaintext, which is described in the next section.

7.4.2 Plaintext Passwords

During our analysis we encountered some apps that send the users' password in plaintext. In fact, we were able to find apps that send the plaintext password via an unencrypted connection. This enables the attacker to easily read the users' credentials by facilitating a Man-in-the-Middle attack. For example, a job platform app on Android from a major Austrian career service sends the users' credentials in plaintext via HTTP. Basically, the request looks like `http://api.obfuscated.at/login?key=user@test.at&secret=geheim1234`. In order not to facilitate potential attacks we obfuscated the URL and we decided not to name the app in this thesis. Furthermore, we have contacted the company and explained them the vulnerability in their API.

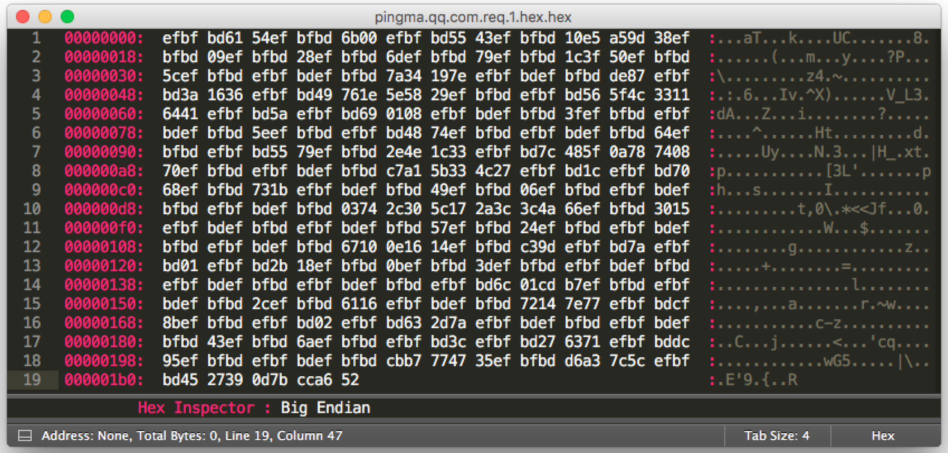
Discussion. We analyzed the aforementioned app and discovered that the username and the users' password is unencrypted in plaintext via HTTP. Considering that most users use the same password for multiple services, this opens a critical vector for attackers. We were able to extract the email address and the password just by employing a Man-in-the-Middle attack on the device under test. In open WiFi networks, such as many coffee shops offer, it is feasible to excerpt the users' passwords. Furthermore, public Internet networks, which are offered on almost every airport in the world, can enable the same attack scenario. In fact, it is crucial to encrypt the connection between the client and the server using a standardized procedure when dealing with users' credentials.

However, other providers try to obfuscate their data in order to make it more difficult to decode the data. Our analyzis has shown that obfuscation is not the same as security. It just increases the steps needed to retrieve the critical data but does provide any security at all. For this reason, we is illustrate this with an example in the next section.

7.4.3 Proprietary Format & Obfuscated Data

During our analysis with the *Timeline* plugin, which is described in Section 5.5, we spotted some apps that sent obfuscated data to their servers. After a detailed analysis, we have discovered that some of them have used proprietary binary formats for transferring data. Interestingly, an Android app that installs a service in the background running with administrator rights, sends most of its data to China. Furthermore, the data is obfuscated by using a proprietary binary format. In fact, we do not know what exactly is sent to the Chinese servers. Moreover, the app is asking for GPS location permission when it has been installed. One example for such an proprietary encoded data that is transferred to Chinese servers is illustrated in Figure 7.3.

There are also tracking frameworks that use proprietary encoding for transmitting their data. One example is the *Crashlytics* iOS SDK. We discovered that this crash tracking service uses a custom encoded and compressed binary with their proprietary *vnd.crashlytics.ios.events* type.



```
00000000:  efbf bd61 54ef bfbd 6b00 efbf bd55 43ef bfbd 10e5 a59d 38ef  :...aT...k....UC.......8.
00000018:  bfbd 09ef bfbd 28ef bfbd 6def bfbd 79ef bfbd 1c3f 50ef bfbd  :......(...m...y....?P...
00000030:  5cef bfbd efbf bdef bfbd 7a34 197e efbf bdef bfbd de87 efbf  :\.........z4.~..........
00000048:  bd3a 1636 efbf bd49 761e 5e58 29ef bfbd efbf bd56 5f4c 3311  :.:.6...Iv.^X)......V_L3.
00000060:  6441 efbf bd5a efbf bd69 0108 efbf bdef bfbd 3fef bfbd efbf  :dA...Z...i........?.....
00000078:  bdef bfbd 5eef bfbd efbf bd48 74ef bfbd efbf bdef bfbd 64ef  :....^......Ht.........d.
00000090:  bfbd efbf bd55 79ef bfbd 2e4e 1c33 efbf bd7c 485f 0a78 7408  :.....Uy....N.3...|H_.xt.
000000a8:  70ef bfbd efbf bdef bfbd c7a1 5b33 4c27 efbf bd1c efbf bd70  :p...........[3L'.......p
000000c0:  68ef bfbd 731b efbf bdef bfbd 49ef bfbd 06ef bfbd efbf bdef  :h...s.......I...........
000000d8:  bfbd efbf bdef bfbd 0374 2c30 5c17 2a3c 3c4a 66ef bfbd 3015  :.........t,0\.*<<Jf...0.
000000f0:  efbf bdef bfbd efbf bdef bfbd 57ef bfbd 24ef bfbd efbf bdef  :............W...$.......
00000108:  bfbd efbf bdef bfbd 6710 0e16 14ef bfbd c39d efbf bd7a efbf  :........g............z..
00000120:  bd01 efbf bd2b 18ef bfbd 0bef bfbd 3def bfbd efbf bdef bfbd  :.....+........=.........
00000138:  efbf bdef bfbd efbf bdef bfbd efbf bd6c 01cd b7ef bfbd efbf  :................l.......
00000150:  bdef bfbd 2cef bfbd 6116 efbf bdef bfbd 7214 7e77 efbf bdcf  :....,...a.......r.~w....
00000168:  8bef bfbd efbf bd02 efbf bd63 2d7a efbf bdef bfbd efbf bdef  :...........c-z..........
00000180:  bfbd 43ef bfbd 6aef bfbd efbf bd3c efbf bd27 6371 efbf bddc  :..C...j......<...'cq....
00000198:  95ef bfbd efbf bdef bfbd cbb7 7747 35ef bfbd d6a3 7c5c efbf  :............wG5.....|\..
000001b0:  bd45 2739 0d7b cca6 52                                       :.E'9.{..R

        Hex Inspector : Big Endian

Address: None, Total Bytes: 0, Line 19, Column 47        Tab Size: 4        Hex
```

Figure 7.3: Screenshot of a proprietary encoded request body sent to Chinese servers.

Discussion. As the time of writing, this app that we mentioned above had 50 to 100 million installs on Android. Consequently, there is an interest in understanding what such apps really send to their servers.

The aforementioned example requires the following permissions upon installation: *Device & app history, Identity, Contacts, Location, SMS, Phone, Photos/Media/Files, Storage, Camera, Wifi connection information, Device ID & call information, Download files without notification, force stop other apps, update component usage statistics, retrieve running apps, modify secure system settings,* *read Home settings and shortcuts, write Home settings and shortcuts, receive data from Internet, view network connections, send sticky broadcast, change network connectivity, connect and disconnect from Wi-Fi, disable your screen lock, full network access, close other apps, read terms you added to the dictionary, run at startup, draw over other apps, control vibration, prevent device from sleeping, modify system settings, add words to user-defined dictionary, install shortcuts and uninstall shortcuts*.

Obviously, the app asks for many permissions on the Android system. Our main purpose with the *Timeline* plugin was to give users and researchers the ability to easily understand what an app does. By using the *Location Destination* plugin we were able to discover that the traffic is sent to China. Then, we used the *Timeline* plugin to analyze the network requests and responses in detail. As a consequence, we identified the requests sent to China and analyzed them. Hence, we were able to uncover the proprietary encoded data in the requests. This is critical, because the users or researchers cannot understand without further analysis what the app sends to their servers.

Next steps are to download the APK of the app, decompile it and to understand how it encodes the data. However, we will mention this finding and leave it open for future work, since the purpose of this thesis is the autonomous detection of potential security risks.

7.4.4 Unprotected Personal Data

Another interesting finding was an app that was connecting to a shared *Dropbox* folder and downloaded all resources from there. In fact, the app has used this hidden remote folder as its backend. Hence, it is possible to download, upload or even manipulate the assets of the app. Furthermore, the developer of this app used this *Dropbox* folder for his private and personal files. Thus, we were able to download them and gain insight into his life. For example some personal calendars downloaded from this *Dropbox* folder are shown in Figure 7.4

Discussion. We downloaded the files from the folder and discovered some personal files including calendars containing sensitive information about

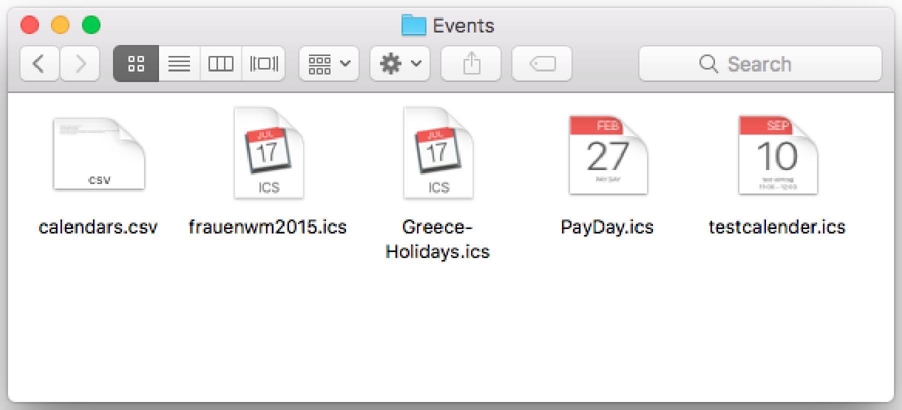


Figure 7.4: Screenshot of an app requesting assets from an unprotected source on Dropbox.

planned vacation in Greece and information about his salary. Hence, we would have been able to compromise those files and manipulate them to whatever we want. Security and privacy in the Internet are serious topics that require extensive knowledge about all stakeholders and the environment in which a system is used.

7.5 Usage of Certificate Pinning

Basically, by using Certificate Pinning, the client verifies the authenticity of the server's public key by pinning a trusted and known certificate. This is described further in Section 3.2. We have defined apps that used certificate pinning as out of scope for this thesis, since the main purpose of this thesis was to automatically analyze the network traffic. Hence, this is not possible with apps that facilitate certificate pinning. However, we were able to observe that the same apps from the exactly same company implement certificate pinning on one platform but not on the other. For instance, the Facebook Pages app implements certificate pinning on iOS but not on

Android, whereby the Facebook Messenger app uses certificate pinning on both platforms.

Discussion. In fact, we did not find this result not in our analysis phase but while we were capturing the network traffic of all apps. We observed that some apps were not able to connect to their servers while the connection through our SSL proxy was established. Basically, these apps used pinned certificates in order to establish the trust with their servers. Since we have injected a trusted but self signed certificate the apps have refused to connect. During our collecting phase we only detected a few apps that have used certificate pinning. However, the aforementioned difference within the Facebook apps might be because of different release cycles on the iOS and on the Android platform.

8 Future Work

While this thesis has discovered and demonstrated significant privacy implications of deployed tracking frameworks and security issues in mobile apps, the results described in Chapter 7 point to several interesting directions for future work. This chapter presents some of these directions.

8.1 Apps implementing Certificate Pinning

The web application developed for this thesis, automatically analyzes captured network dumps, whereby these network dumps must be captured by using Burp Suite as an SSL proxy between the mobile device and their servers. Hence, certificate pinning prevents the client from communicating with their servers because of our deployed SSL proxy. Thus, we defined apps with certificate pinning as *out of scope* for this thesis and omitted them in our capturing phase.

Extending the analysis and its results with captured traffic from apps that facilitate certificate pinning, is definitely interesting and can outpoint the results from this master thesis.

8.2 Offline Apps

Mobile apps that solely operate offline are rare but still exist. The same as described in the previous section applies for apps without any network traffic. This thesis is based on the analysis of the captured network traffic. Thus, apps without network transmission have been considered as *out of scope* of this thesis.

8.3 Analyze Methods for other Tracking Libraries

The automated analysis of the tracking and advertisement frameworks includes an automatic analysis of the network traffic sent to their hosts. This is implemented in *Detailed Analysis of Tracking Libraries* plugin described in Section 5.2. Considering the amount of 64 tracking libraries that our plugins can detect we have not implemented the *analyze* method for all the supported frameworks. Thus, the plugin can be extended for supporting further tracking frameworks in order to provide an automated decoding of the content.

For instance, the crash debugging and tracking framework *Crashlytics* uses a custom encoded binary for transmitting the tracked data to their servers. They use a content type named *Application/vnd.crashlytics.ios.events*. This is a proprietary format that *Crashlytics* has defined. Since it was not obvious how to decode the binary, we have defined this proprietary format as *out of scope* for this thesis. However, we consider decoding custom binaries as a direction as possible future basis for work.

8.4 In-House Tracking

This thesis focused on tracking libraries that were offered as a service or product from tracking and advertisement providers. During our analysis we also found tracking requests that were targeted to in-house servers from particular apps. Big companies, such as Amazon or Uber are implementing their own tracking mechanisms in order to keep the data in-house. The tracked data from in-house solutions is definitely worth to be considered. Hence, we propose this as a direction for future work. This thesis primary goal was the automated analysis which cannot be applied to in-house solutions.

8.5 Increased Sample Size

Another possible direction for future work is the sample size. We collected the network traffic from the first 10 or 20 apps from each category on the Android PlayStore and the iOS AppStore. Thus, we gathered 440 apps in total, which was our sample size. By increasing the amount of tracked apps the results can become more representative. Hence, we propose this for future work. Furthermore, considering the implications of the ranking based on the user's AppStore region we also propose to evaluate the results from different AppStore and PlayStore regions to compare with our results.

9 Conclusion

We have argued that most mobile applications track sensitive information about their users and send them to tracking servers. Moreover, we have developed a framework and five plugins in order to evaluate and to confirm our hypothesis. For this reason, we collected the network traffic of 440 apps from the iOS and the Android platform and with the results of the automated analysis of our plugins we were able to validate our assumptions. In particular, we were able to prove that most apps on the market facilitate tracking or advertisement frameworks and furthermore, many of them use the exactly same unique device identifier for transferring their data. This enables the providers to connect the data from different apps together and to create accurate personas. Moreover, the shared identifier enables different providers to exchange their data, which implies that it is possible to derive real entities from the data. This has serious impact on the privacy of each user.

This thesis confirms previous findings and contributes to the understanding of security and privacy in mobile apps. Our results have shown that tracking enables advertisers to personalize their advertisements in a way that can have an impact to our society. When the advertisement providers know about our personal preferences, details about our job and income, or where we are living, just by tracking behavioral data from the mobile apps we use every day. Moreover, the results we were able to obtain from the automated analysis of the network dumps and the manual inspection with the *Timeline* plugin have shown a lack of security awareness from the developers side and a missing privacy awareness from the users side.

Most people are not aware of the amount of data that is being tracked by the apps we are using every day. Especially our findings in the *Games* category regarding the number of tracking frameworks and the usage of unique device identifiers, were immensely significant. Every single app in this category contained at least one tracking framework. Moreover, most mobile applications have used more than one tracking library. In fact, our results in the *Games* category show that more than 85% of the apps in this category were using the exactly same unique device identifier within their traffic. Hence, like aforementioned, they are able to connect the collected data and define persons.

George Orwell said in his book *"If you want to keep a secret, you must also hide it from yourself."* [45]. Our analysis has shown that nowadays it is not about the data we actively expose to others, it is more the data that is tracked from our behavior that matters. In 2014, *Flurry* one of the market leaders in mobile app tracking, has been acquired by *Yahoo* for 200 Million US Dollars, which shows how important and worthy tracking data companies have become. Companies and tracking providers have already realized that behavioral data is valuable. On the other hand, we as users, should also realize that, most of the time when we are using a free service from a company, we are actually the products and not the service. For that reason, data really do matter.

Bibliography

[1] A. Cortesi and M. Hils, "How MitM Proxy works," 2014.

[2] C. Shin and A. Dey, "Automatically Detecting Problematic Use of Smartphones," pp. 335–344, 2013.

[3] M. Tsavli, P. S. Efraimidis, V. Katos, and L. Mitrou, "Reengineering the user: privacy concerns about personal data on smartphones," *Information & Computer Security*, vol. 23, no. 4, pp. 394–405, 2015.

[4] Google, "Personalized Advertising."

[5] PrivacyRights, "Privacy in the Age of Smartphones."

[6] Flurry, "Reach Your Target Audience with Flurry Personas," tech. rep.

[7] Apple, "Answers to your questions about Apple and security," tech. rep., Apple.

[8] T. Danovy, "The Global Smartphone Report," tech. rep., Business Insider Deutschland, 2015.

[9] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," *Proc. 5th ACM conference on Security and Privacy in Wireless and Mobile Networks*, vol. 067, no. Section 2, pp. 101–112, 2012.

[10] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices - SPSM '11*, pp. 3 – 14, 2011.

[11] T. Chen, I. Ullah, M. A. Kaafar, and R. Boreli, "Information leakage through mobile analytics services," *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications - HotMobile '14*, pp. 1–6, 2014.

[12] C. A. Eldering, "Advertisement auction system," 2001.

[13] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. L. Traon, D. Octeau, and P. Mcdaniel, "FlowDroid : Precise Context , Flow , Field , Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps," *PLDI '14 Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 259–269, 2014.

[14] A. Arora, S. Garg, and S. K. Peddoju, "Malware Detection Using Network Traffic Analysis in Android Based Mobile Devices," *Next Generation Mobile Apps, Services and Technologies (NGMAST), 2014 Eighth International Conference on*, pp. 66–71, 2014.

[15] S. Schrittwieser, P. Frühwirt, P. Kieseberg, M. Leithner, M. Mulazzani, M. Huber, and E. Weippl, "Guess who's texting you? evaluating the security of smartphone messaging applications," *Proceedings of the 19th annual symposium on network and distributed system security*, p. 9, 2012.

[16] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and S. Wang, "AppIntent : Analyzing Sensitive Data Transmission in Android for Privacy Leakage Detection," *CCS '13 Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 1043–1054, 2013.

[17] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," *Osdi '10*, vol. 49, pp. 1–6, 2010.

[18] Juniper Networks, "SSL Proxy Overview," 2012.

[19] E. De La Hoz, G. Cochrane, J. M. Moreira-Lemus, R. Paez-Reyes, I. Marsa-Maestre, and B. Alarcos, "Detecting and defeating advanced man-in-the-middle attacks against TLS," *International Conference on Cyber Conflict, CYCON*, pp. 209–221, 2014.

[20] R. Shirey, "Internet Security Glossary," *Rfc 2828*, pp. 1–212, 2000.

[21] R. Oppliger, R. Hauser, and D. Basin, "SSL/TLS session-aware user authentication - Or how to effectively thwart the man-in-the-middle," *Computer Communications*, vol. 29, no. 12, pp. 2238–2246, 2006.

[22] FPF Application Privacy, "Device Identifiers," 2012.

[23] Apple, "UIDevice Class Reference: Identifier For Vendor."

[24] AVG, "Apple iOS 7 puts an end to unique device IDs," 2013.

[25] Apple, "ASIdentifierManager Class Reference: Advertising Identifier," 2016.

[26] G. Sterling, "Google Replacing "Android ID" with "Advertising ID" similar to Apple's IDFA," 2013.

[27] Google Services, "Settings Secure: Android Identifier," 2016.

[28] Google, "Platform Versions," 2016.

[29] Pew Research, "Android apps now ask for over 200 kinds of permissions," 2015.

[30] S. J. Tipton, D. J. White Ii, C. Sershon, and Y. B. Choi, "iOS Security and Privacy: Authentication Methods, Permissions, and Potential Pitfalls with Touch ID," *International Journal of Computer and Information Technology*, vol. 03, no. 03, pp. 2279–764, 2014.

[31] S. Ranger, "iOS versus Android. Apple App Store versus Google Play: Here comes the next battle in the app wars — ZDNet," 2015.

[32] N. Ingraham, "Apple's App Store has passed 100 billion app downloads," 2015.

[33] N. Seriot, "iPhone privacy," *Black Hat DC*, p. 30, 2010.

[34] D. Kaplan, "Google using custom malware scanner for Android apps," *SC Magazine*, 2012.

[35] Industrial Safety and Security Source, "Chemical Safety Incidents Google Play Malicious Apps Up 400%," 2014.

[36] R. Kuć and M. Rogoziński, *Mastering ElasticSearch*. Packt Publishing, first edit ed., 2013.

[37] ElasticSearch, "Query DSL."

[38] Yahoo, "https://developer.yahoo.com/advertise/."

[39] Google, "https://www.doubleclickbygoogle.com/solutions/measurement/marketers/."

[40] Flurry, "http://www.flurry.com/solutions/advertisers/brands."

[41] The Verge, "Windows Phone Market Share."

[42] Port Swigger, "Burp Suite."

[43] Port Swigger, "Burp Suite Scanner."

[44] OWASP, "OWASP Top 10."

[45] G. Orwell, *1984*. 1949.

Appendix

					Results															
	# of Apps		Unique Identifiers found		Same ID used by # of apps (out of # of Apps)															
Category	iOS	Android	iOS	Android	iOS								Android							
Business	10	10	**16**	**19**	**5**	**50%**	-	**-**	-	**-**	-	**-**	**4**	**40%**	2	**20%**	2	**20%**	2	**20%**
Education	10	10	**39**	**9**	**7**	**70%**	2	**20%**	-	**-**	-	**-**	**4**	**40%**	-	**-**	-	**-**	-	**-**
Entertainment	10	10	**28**	**20**	**7**	**70%**	-	**-**	-	**-**	-	**-**	**6**	**60%**	-	**-**	-	**-**	-	**-**
Finance	10	10	**11**	**14**	**6**	**60%**	-	**-**	-	**-**	-	**-**	**4**	**40%**	-	**-**	-	**-**	-	**-**
Food & Drink	10	-	**18**	**-**	**5**	**50%**	-	**-**	-	**-**	-	**-**		**-**	-	**-**	-	**-**	-	**-**
Games	20	20	**98**	**45**	**17**	**85%**	3	**15%**	2	**10%**	2	**10%**	**18**	**90%**	2	**10%**	2	**10%**		
Heatlh & Fitness	10	10	**20**	**20**	**6**	**60%**	2	**20%**	2	**20%**	2	**20%**	**7**	**70%**	-	**-**	-	**-**	-	**-**
Kids	10	-	**21**	**-**	**8**	**80%**	3	**30%**	2	**20%**				**-**	-	**-**	-	**-**	-	**-**
Lifestyle	10	10	**29**	**14**	**5**	**50%**	-	**-**	-	**-**	-	**-**	**6**	**60%**	-	**-**	-	**-**	-	**-**
Medical	10	10	**19**	**20**	**3**	**30%**	-	**-**	-	**-**	-	**-**	**4**	**40%**	-	**-**	-	**-**	-	**-**
Music	10	10	**35**	**17**	**7**	**70%**	-	**-**	-	**-**	-	**-**	**5**	**50%**	-	**-**	-	**-**	-	**-**
Navigation	10	-	**28**	**-**	**4**	**40%**	-	**-**	-	**-**	-	**-**		**-**	-	**-**	-	**-**	-	**-**
News	10	-	**14**	**-**	**5**	**50%**	-	**-**	-	**-**	-	**-**		**-**	-	**-**	-	**-**	-	**-**
Photo & Video	10	10	**52**	**11**	**7**	**70%**	-	**-**	-	**-**	-	**-**	**6**	**60%**	-	**-**	-	**-**	-	**-**
Productivity	10	10	**20**	**13**	**5**	**50%**	3	**30%**	2	**20%**			**5**	**50%**	-	**-**	-	**-**	-	**-**
Reference	10	-	**24**		**5**	**50%**	-	**-**	-	**-**	-	**-**		**-**	-	**-**	-	**-**	-	**-**
Shopping	10	10	**30**	**24**	**8**	**80%**	-	**-**	-	**-**	-	**-**	**8**	**80%**	-	**-**	-	**-**	-	**-**
Social Networking	10	10	**19**	**25**	**8**	**80%**	-	**-**	-	**-**	-	**-**	**9**	**90%**	-	**-**	-	**-**	-	**-**
Sports	10	10	**24**	**8**	**5**	**50%**	-	**-**	-	**-**	-	**-**	**2**	**20%**	-	**-**	-	**-**	-	**-**
Travel	10	10	**39**	**19**	**4**	**40%**	-	**-**	-	**-**	-	**-**	**5**	**50%**	-	**-**	-	**-**	-	**-**
Utilities	10	10	**24**	**4**	**5**	**50%**	-	**-**	-	**-**	-	**-**	**5**	**50%**	-	**-**	-	**-**	-	**-**
Weather	10	10	**22**	**5**	**6**	**60%**	2	**20%**	2	**20%**			**3**	**30%**	-	**-**	-	**-**	-	**-**
Communication	-	10	**-**	**13**	**-**	**-**	-	**-**	-	**-**	-	**-**	**4**	**40%**	-	**-**	-	**-**	-	**-**
Media & Video	-	10	**-**	**15**	**-**	**-**	-	**-**	-	**-**	-	**-**	**2**	**20%**	-	**-**	-	**-**	-	**-**
Transportation	-	10	**-**	**1**	**-**	**-**	-	**-**	-	**-**	-	**-**	**1**	**10%**	-	**-**	-	**-**	-	**-**
	230	**210**				**59%**		**23%**		**18%**		**15%**		**50%**		**15%**		**15%**		**20%**

	# of Apps		# of Apps which send the majority of their Network Requests to Hosts in this Country																								
			iOS													Android											
Category	iOS	Android	US	AT	NL	DE	IE	FR	PL	CZ	CN	DK	HR	CA	GB	US	AT	NL	DE	IE	CZ	HR	CN	SE	CA	BG	GB
Business	10	10	6	2		1		1								7	1		1	1							
Education	10	10	3	1	1		4	1								8	1		1								
Entertainment	10	10	8		1		1									6		3		1							
Finance	10	10	5	2	1		2									3	4	1		1						1	
Food & Drink	10	-	6	2	1										1												
Games	20	20	18		1									1		20											
Heatlh & Fitness	10	10	5			4	1									8			2								
Kids	10	-	8			1						1															
Lifestyle	10	10	5	2	1			1	1							5	2		2	1							
Medical	10	10	4		1	5										9			1								
Music	10	10	5			4									1	7	1							1	1		
Navigation	10	-	4	4	1		1																				
News	10	-	1	7	1	1																					
Photo & Video	10	10	8				1				1					9				1							
Productivity	10	10	5		1	3	1									7							3				
Reference	10	-	10																								
Shopping	10	10	7	1		1							1			6	1		1			1					1
Social Networking	10	10	7			1	2									7			1	2							
Sports	10	10	4	2	1	1	1			1						2	3	1	1	2	1						
Travel	10	10	3	2	4	1										5	3	2									
Utilities	10	10	5	2	1		1								1	5	1			4							
Weather	10	10	4	4	1	1										7	2	1									
Communication	-	10														6	1	1	1	1							
Media & Video	-	10														5	1	1			1		2				
Transportation	-	10														3	4		3								
	230	210	131	31	17	24	15	3	1	1	1	1	1	1	3	135	25	10	14	14	2	1	5	1	1	1	1