Network Traffic Analysis of Android Applications

NF4Droid - A Network Forensics Tool for Android Security Experts

Lumper Christian, BSc

Network Traffic Analysis of Android Applications

NF4Droid - A Network Forensics Tool for Android Security Experts

Master's Thesis

at

Graz University of Technology

submitted by

Lumper Christian, BSc

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology
A-8010 Graz, Austria

October 15, 2012

Advisor: Univ.-Prof. M.Sc. Ph.D. Bloem Roderick
Co-Advisor: Dipl.-Ing. Dr.techn. Teufl Peter

Netzwerkverkehr Analyse von Android Applikationen

NF4Droid - Ein Netzwerk-Forensik Tool für Android Sicherheitsexperten

Masterarbeit

an der

Technischen Universität Graz

vorgelegt von

Lumper Christian, BSc

Institut für Angewandte Informationsverarbeitung und Kommunikationstechnologie (IAIK),
Technische Universität Graz
A-8010 Graz

15. Oktober 2012

Diese Arbeit ist in englischer Sprache verfasst.

Begutachter:	Univ.-Prof. M.Sc. Ph.D. Bloem Roderick
Mitbetreuender Assistent:	Dipl.-Ing. Dr.techn. Teufl Peter

Abstract

Modern mobile devices in combination with sophisticated mobile platforms opened up new possibilities for the development of mobile applications (apps). However, new threat types did arise with the new capabilities. Apps which harm the privacy of the user or even show malicious behaviour (malware) were created. This required the development of app analysis and malware detection methods.

This thesis outlines threat types for mobile platforms and describes the integrated security measures of the platforms. Furthermore, it addresses the issue that the Android platform is one of the primary targets for malware. Subsequently, it describes current Android app analysis and malware detection methods and their requirements on the analysis environment.

Since less research on the analysis of the network from Android apps could be found, it highlights the great potential to reveal possible unwanted or even malicious behaviour of apps through the analysis of the network traffic. Consequently, the tool NF4Droid (Network Forensics For Android), specialised for the analysis of network traffic captured from Android apps, was developed in the scope of this thesis. NF4Droid provides rich presentations and visualisations of the network traffic and applies in-depth analysis for the identification of data exposure, what should help to understand the behaviour of apps.

To study the network traffic of Android apps and to evaluate the capabilities of NF4Droid the top 50 free apps from Google Play and some known malicious apps were analysed. The analysis allowed to gain more knowledge about the general communication behaviour apps and did yield to the result that both apps commonly expose information to advertising companies, frequently without notice of the user.

Keywords: Network Traffic Analysis, Mobile Platform Threat Types, Android, App Analysis Methods, Malware Detection, Grayware, Privacy, Mobile Advertising

Kurzfassung

Mobile Endgeräte und die dazugehörigen Plattformen haben weitreichende Möglichkeiten für die Entwicklung mobiler Anwendungen (Apps) eröffnet. Mit den neuen Möglichkeiten sind aber auch neue Gefahren entstanden. Apps sind entwickelt worden, die die Privatsphäre des Benutzers missachten oder bösartiges Verhalten zeigen (Malware). Daher ist es notwendig App Analyse- und Malware Erkennungsmethoden zu finden.

Im Rahmen dieser Arbeit werden die Sicherheitsmaßnahmen und Bedrohungstypen für mobile Plattformen beschrieben. Weiters wird erläutert warum die Android Platform eines der Hauptziele für Malware ist. Nachfolgend werden aktuelle Android App Analyse- und Malware Erkennungsmethoden und deren Anforderungen an die Testumgebung besprochen.

Da wenig Forschung in Richtung der Analyse des Netzwerkverkehrs von Android Apps betrieben worden ist, beschreibt die Arbeit das Potential dieser Methode für die Erkennung von unerwünschtem oder sogar bösartigem Verhalten von Apps. In weiterer Folge ist im Rahmen dieser Arbeit das Tool NF4Droid (Network Forensics For Android) entwickelt worden, das auf die Analyse des Netzwerkverkehrs von Android Apps spezialisiert ist. Das Tool bietet umfangreiche Darstellungsformen und Visualisierungen für den Netzwerkverkehr und wendet eine umfangreiche Analyse zur Erkennung von Informationen im Netzwerkverkehr an. Damit soll NF4Droid Experten helfen das Verhalten von Apps besser zu verstehen.

Zur Auswertung des Netzwerkverkehrs von Android Apps und zur Evaluierung von NF4Droid sind die Top 50 der kostenlosen Apps von Google Play zusammen mit einigen als Malware bekannten Apps getestet worden. Die Analyse ermöglicht das Sammeln allgemeiner Erkenntnisse über das Kommunikationsverhalten von Apps und hat zu dem Resultat geführt, dass Apps häufig Informationen an Werbefirmen weitergeben - meist ohne das Wissen des Benutzers.

Schlüsselwörter: Netzwerkverkehr Analyse, Mobile Plattformen Bedrohungstypen, Android, App Analyse Methoden, Malware Erkennung, Grayware, Privatsphäre, Mobile Werbung

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.

Eidesstattliche Erklärung

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 entnommene Stellen als solche kenntlich gemacht habe.

Signature Value	OHTtbDuiHDGD/TWG61wLEBQOp2Juxa7PRpzd5T7L2MG02x84KHTSRbsYCC4YLMYvG3gW8r4+My2f1MQQoZGQEA==	
SIGNATURE INFORMATION	**Signatory**	Christian Lumper
	Issuer-Certificate	CN=a-sign-Premium-Sig-02,OU=a-sign-Premium-Sig-02,O=A-Trust Ges. f. Sicherheitssysteme im elektr. Datenverkehr GmbH,C=AT
	Serial-No.	656614
	Method	urn:pdfsigfilter:bka.gv.at:text:v1.2.0
	Parameter	etsi-moc-1.1@9c052507
Verification	Signature verification at: http://www.signature-verification.gv.at	
Note	This document is signed with a qualified electronic signature. According to section 4 para 1 of the Signature Act it in principle is legally equivalent to an handwritten signature.	
Date/Time-UTC	2012-10-14T23:05:44Z	

Acknowledgements

First of all, I would like to thank my co-advisor, Peter Teufl, from Graz University of Technology for proposing this thesis and his dedicated help throughout the whole project. Especially for his openness and flexibility to let me finish my thesis from my hometown.

Moreover, I would like to thank the Institute for Applied Information Processing and Communications (IAIK) from Graz University of Technology and my supervisor, Roderick Bloem, for giving me the chance to do this work at the institute.

Many thanks to Bernhard Mentler for proof reading this document. I would also like to thank David Gstir, Matthias Kegele and Mathias Winder for their inspiration and help over the whole years of studying.

Special thanks go to my girlfriend, Martina Groder, who has always been there for me. Finally I would like to thank my father, Alfred Lumper, and the rest of my family (Heidi and Manfred) for their long-standing aid over the course of my academic studies.

Lumper Christian
Vorderhornbach, Austria, October 2012

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter will give an overview of the motivation for this work and present some background information about mobile platforms their threats and security measures, and briefly describe application analysis methods. Additionally, we describe our contribution to the subject of network traffic analysis and outline the further structure of the thesis.

1.1 Motivation

The technical features of modern mobile devices and the development of sophisticated mobile platforms led to a significant increase in the sale of smartphones and tablets over the past years [52, 96]. Besides technical improvements such as high-resolution touchscreens and location services, the development of mobile platforms with a rich feature set and the extendibility through third-party applications mainly contributed to the success of smartphone, as well as, tablets. According to analysis by Gartner Inc. [95] in May 2012 especially the mobile platforms Android from Google [98] with a market share of 56.1% and iOS from Apple [34] with 22.9% could register a success over previously established platforms like Symbian [178] (8.6%) and the BlackBerry OS from Research In Motion (RIM) [163] (6.9%). Although Windows Phone from Microsoft [142] has currently a low market share (1.9%) predictions indicate that Windows Phone will be able to catch up till 2013 as the third biggest mobile platform [94].

Modern mobile platforms offer developers Software Development Kits (SDKs) which support the creation of applications and allow to easily use the technical features provided by the devices like the Global Positioning System (GPS) or the camera and access data like contacts or short messages. Additionally, the platform providers offer digital-distribution channels like the Apple App Store [35], Windows Phone Marketplace [140] or Google Play [99] (formerly Android Market) which allow the developers to easily present, distribute and sell their applications. This comprehensive environment and the big market pushed the development of applications for mobile platforms. This led to an impressive amount of available applications, with each of the two biggest official markets (Apple App Store and Google Play) offering more than 600,000 applications [125].

Unfortunately, the popularity of the mobile platforms and the presence of sensitive information on the devices make the platforms an attractive target for developers of malicious applications (malware) and consequently new security threats arise. Malware might harm the privacy of the user by exposing private data like contacts or reveal sensitive information like the location. Moreover, attackers utilise their applications to spy on the user or contact premium services, which cause financial charges. Besides, free applications commonly share various sensitive information with advertising companies without the knowledge of the user. The rapid growth of mobile malware in the recent years, with an increase of 155% in 2011 and solely 46.7% of the overall malware target for Android, emphasises the importance of measures to identify possible malicious applications [128].

1.2 Background

Mobile platforms take various security measures to protect their system and prevent the development and distribution of malware. Platforms use established security architectures adapted and extended for the use case on mobile devices. Extensions include methods such as sandboxing, application signing and permission systems for the access of sensitive data and hardware features. Furthermore, platforms like iOS from Apple rely on manual approval processes at the distribution channel to identify malicious applications [37]. On the contrary, Android with a different security architecture and a more open philosophy does not apply such approval methods. Android uses a tool called Bouncer for the automated analysis of applications on Google Play and relies on the community to identify and report malicious applications [114]. All the mentioned security measures try to limit the risk of malware but can not guarantee a total protection. As a consequence mobile platforms integrate methods which allow to remotely remove malicious applications from the devices [148, 164].

Additionally to applications showing direct malicious behaviour there exists a variety of applications which show possible unwanted but not yet malicious behaviour. However, those are not covered by this protection methods. For example, a lot of free applications use information like unique identifiers and the device location for advertising purposes without the direct knowledge of the user.

Hence, different methods for the analysis of applications and the detection of malware exist and are continuously developed by researchers with the aim to limit the impact of malicious applications. The distinct methods have, however, different requirements on the analysis environment. In some cases they can only be deployed on rooted devices or require modifications of the system, whereas others can be run on standard devices or even emulated environments. Furthermore, the methods very in the demand of user interaction and the possibility to be automated. In addition, further requirements influence the applicability and the selection of adequate analysis methods.

Especially for Android a variety of analysis methods have been developed by the community, on the one hand because Android is currently the main target of malware [128], on the other hand because the openness of the system offers researches many possibilities for the development and deployment of analysis methods. In the following we briefly describe some of the established analysis and detection methods:

The user is one of the first members in the chain of the malware detection. An experienced user might identify, if the required permission of an application are reasonable and match the context of the application. The unexpected behaviour, like the crash of an application, might raise the suspicion of the user, but without any further tools the possibilities are quite limited.

Mobile security applications offer functionality, known from desktop computer antivirus software, to scan the system for malicious applications. The capabilities of such applications are however limited by the mobile system and their system resources. As a consequence they can usually just identify known malware.

Static analysis helps to identify new malware by analysing the source code of an application or the reverse engineered binary installation package of the application. The methods include simple signature-based malware detection and range over call-graph analysis up to the simulation of execution chains. The static analysis does not rely on a device or emulator and usually no interaction of the user is required. However, methods for obfuscating the source code can make it hard to restore the actual source code and reproduce the behaviour.

The *dynamic analysis* monitors the execution of an application on a physical or emulated device. It is usually implemented by either instrumenting the application code or executing the application in a modified environment. Although only the monitored behaviour can be analysed, the advantage is that detailed information about the current execution path and the according system status is available. For the effective dynamic analysis the executed application must be provided with sufficient input by either the user or automatically generated input.

The *metadata analysis* tries to find malicious applications by applying knowledge discovery techniques on the information provided on the markets to describe the applications. Such information includes the required permissions, the textual description of the application, the assigned category or the user rating. The analysis can be considered as high-level method since it does not require the application package and there is no need to execute the application on a real or emulated device.

The process of capturing and *analysing the network traffic* between applications and their contacted servers does deliver a broad range of information about what data leaves the device. The traffic might include various sensitive data available on the mobile platform like unique device identifiers, the location or the phone number. The examination of this information could help to detect leakage of private data without the approval and notice of the user. Moreover, malicious activities of applications might be observable through the network traffic.

Besides established and well researched methods, like the static- and dynamic analysis, less research can be found on the analysis of the network traffic produced by applications. The Wall Street Journal [183] in cooperation with the technology consultant David Campbell published an article, where they analysed 101 popular iOS and Android applications relating to data privacy. While the report reveals interesting details about the misuse of private data by applications for advertisement purposes, there is no information given about the exact testing method and the results are limited to the tested applications. Fulton [92] presented on the DEF CON 19 [67] his general work on the analysis of the network traffic of Android applications and points out some examples of private data exposed by applications.

Since the network is one of the main interfaces to the outerworld of today's devices, it has great potential to reveal the behaviour of applications. With various applicable network traffic capturing techniques, the analysis method is independent of the underlying test environment and can be deployed, regardless if a standard, rooted or even emulated device is used. Furthermore, it only requires the installed application and it does not matter if manual user interaction or simulated user input is utilised for testing.

With the automated network traffic analysis we could get a deeper insight about the general functioning of applications and answer question like, how much and what data is transmitted to which servers. Furthermore, we could unveil what protocols are deployed by the application and if secure communication channels are used. Besides this general obtainable information, the detailed examination of the transmitted data could reveal the misuse of sensitive data and the exposure of private information, and thus help to identify undesired or even malicious behaviour of applications.

1.3 Contribution

For the purpose of this thesis we developed a tool called NF4Droid which allows to analyse the network traffic of Android applications. NF4Droid offers functionality to manage different traffic captures of applications and offers extensive features for the visualisation of the network traffic. The visualisations should give the user deeper insight about what connection are established by the applications and reveal the usage of the different network protocols and secure communication. Furthermore, it allows to easily identify the location of the contacted servers. The visualisation of the network traffic over time shows at which point an application does establish connections and how much data is transferred (as demonstrated in Figure 1.1). This summarised representation of the traffic information enables the user to better understand the behaviour of the application.

Besides the more general presentations and visualisations of the network traffic, offers NF4Droid specialised capabilities to identify the exposure of sensitive information related to Android as the underlying mobile platform. The network traffic is inspected by NF4Droid for the leakage of information such as unique identifiers of the device, the location, the phone number or the contacts. The possible exposure of data is as well visualised in a manner that should make it easy for the user to identify the suspicious actions of applications as shown in Figure 1.1.

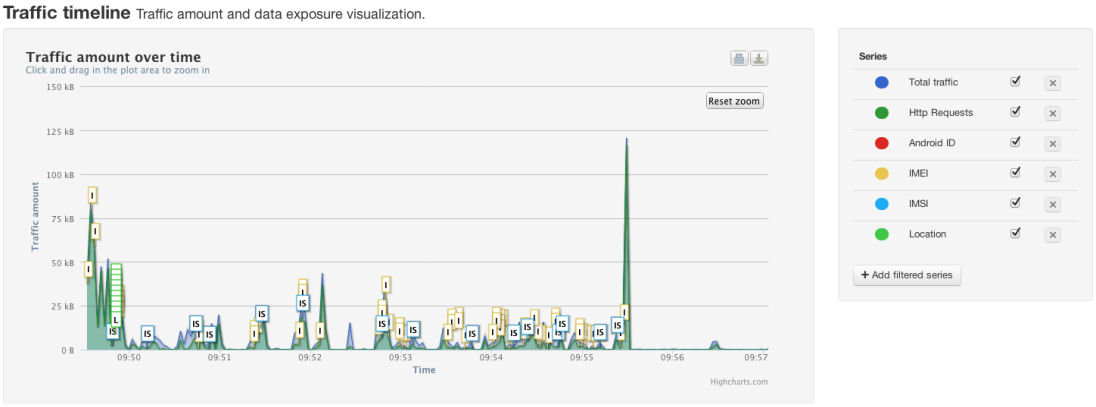

Figure 1.1: Screenshot of the "Traffic Timeline" section in NF4Droid, presenting the network traffic amount over the time with flag markers denoting exposure of information. The given visualisation presents the results for the "Cut the Rope - Free" application indicating the extensive transmission of sensitive data for advertising purposes.

The overall information about the network traffic of an application presented by NF4Droid should assist expert users in the process of identifying applications which disregard the privacy of the user and help to reveal malicious behaviour of applications.

To test and demonstrate the capabilities of NF4Droid we analysed the network traffic of the top 50 free applications on Google Play and moreover from known malicious applications. The results show that NF4Droid could reveal application with aggressive advertisement practices which harm the privacy of the user. Furthermore, the tool proofs its partial ability to identify malicious action of malware utilising the network.

1.4 Outline

The remainder of this thesis is structured as follows. Chapter 2 gives background information about mobile platforms, applications and malware. We present an overview of the different threat types for mobile platforms, including samples of malware, and identify motives for the development of malicious applications. Next, the security measures of the mobile platforms and their distribution channels are highlighted. Furthermore, we point out reasons for attackers to target malware for Android and emphasise why this work is focused on Android.

Chapter 3 presents existing methods for the analysis of Android applications and the detection of malware. The chapter begins with the characterisation of the underlying environment of the analysis methods. Next, the role of the user in the detection process and the use of mobile security applications is described. Furthermore, work related to static, dynamic and metadata analysis is presented in more detail. Finally, the currently existing work on the analysis of the network traffic is presented and we discuss advantages and disadvantages of all analysis methods.

Chapter 4 highlights the problem that currently established methods for the analysis of network traffic are not well suited for the inspection of the traffic from Android applications. We point out which information general network traffic analysis could unveil and describe the potential of the in-depth analysis for the detection of data exposure. Finally, we enumerate threat types which might be detectable with network traffic analysis.

Chapter 5 introduces the chosen approach for the analysis of the network traffic using the developed web application NF4Droid. First, some information about the process of capturing the network traffic

is given. Next, the data processing steps involving the data import, the further processing and the in-depth analysis for data exposure are described. Finally, the pursued methods for the visualisation and representation of the network traffic are presented.

Chapter 6 reveals implementation details of the developed web application NF4Droid. Including the parsing of the traffic capture, the storage of the gathered information and the designed data model behind it. Furthermore, we present some details about the realisation of the web application and refer to the libraries used for the implementation of the user interface and the visualisation of the network traffic.

In Chapter 7 a case study, were we analysed the network traffic of the top 50 free applications from Google Play and a set of known malicious applications with the developed tool NF4Droid, is presented. This chapter demonstrates the capabilities of NF4Droid and reveals some interesting findings related to the privacy of the user.

In Chapter 8 we conclude over the whole work presented in this thesis and discuss the results. We finish the thesis with a future perspective on the possibilities to extend the capabilities of NF4Droid and the integration into test frameworks. Moreover, we propose some different approaches for the analysis of the network traffic of Android applications.

Chapter 2

Mobile Platforms, Apps and Malware

Modern mobile operating systems like iOS from Apple and Android from Google gained a significant market share over the past years and overtook classical platforms like Symbian or the BlackBerry OS [95]. Similarly, Microsoft try's to catch up on this numbers with the new platform Windows Phone. In contrast to previous established platforms, the new systems focus on usability and offer greater possibilities for the creation of applications by third-party developers. SDKs provided for mobile platforms support developers in the development process and allow to easily create apps that make use of the technical features offered by today's smartphones. Furthermore, digital distribution platforms like the Apple App Store, Windows Phone Marketplace [140] and Google Play (formerly Android Market) allow developers to easily present, distribute and sell their applications.

The comprehensive environment and the big market pushed the development of applications for mobile platforms. However, not only developers are attracted by the opportunities of the new devices, the mobile platforms and their distribution channels, they also arouse the interest of attackers to create malicious applications. Different threat types for the user arise from such malicious applications. For example, some applications harm the user's privacy through exposure of sensitive data, whereas others, can be utilised to spy on the user, still others try to gain financial benefits.

The mobile platforms try to limit the risk of such incidents by taking different security measures including the use of hardware and software security features on the Operating System (OS) level and approval and verification steps at the distribution channels [5, 38]. Although various security measures have been taken, malicious applications exist for all platforms. Reports actually indicate a large increase in mobile malware over the past years utilising even more sophisticated vulnerabilities and mainly targeting the Android platform [128, 136].

Before we go into the details of application analysis and malware detection methods, we want to give the reader a short introduction to the general topic of malicious applications on mobile platforms. Therefore, the next section does point out common threat types and the motives behind them, as well as it describes the security measures taken by the platforms against them. Since this thesis focuses on the analysis of network traffic from Android applications, we consequently highlight some reasons for attackers to target their attacks against the Android platform.

2.1 Threat Types and Motives

Felt et al. [81] identifies three threat types for mobile platforms: *malware*, *spyware* and *grayware*. The classification is based on their distribution method, legality and notice to the user. The paper further gives deeper insight into the current and potential future motives of attackers to develop such applications and describes possible defence mechanisms.

This section will briefly present this categorisation and highlight the most important motives for the

attackers. In this thesis, not only applications are referenced with the term *application*, but also *games*, *widgets* and other kind of *utilities* for mobile platforms. Accordingly, all the mentioned threats even apply to software of this kind.

2.1.1 Malware

Malware is considered to pose a significant security risk to the users' system and information [79]. Developers of malware deceive users to install malicious applications or try to use other system vulnerabilities to gain unauthorised access. Malicious actions include the destructive behaviour to the system or data, the stealing of user credentials or the collection of user information and other sensitive data. Consequently, malware does not provide any legal notice to the user for the actions carried out.

Trojans, worms, botnets and viruses belong to the category malware [80], and in the Section 2.1.4 we present some example and describe they motives for the development of such malicious applications.

Although the expression malware references to the specific threat type described above it is often used as umbrella term to characterise even the following two threats.

2.1.2 Personal Spyware

In contrast to regular malware with spying capabilities, which does not proclaim its spying functionality and hides it behaviour in any way, dose *personal spyware* actively advertise its ability to spy a person and requires physical access to the device for installation. However, personal spyware hides itself as well from the user while collecting personal information like the call history, text messages or tracking the location. Various threat types arise from the capabilities of such applications which are described in Section 2.1.4.6. Unlike for malware, the information from personal spyware is, however, gathered for the person which initialised the spying, rather than the application developer.

For the Android platform various such applications (e.g. MobiStealth [145], Spy Control [187]) are available on Google Play and advertise themselves as spyware. A common but controversial use case of such applications is the spying on the significant other or the parental control. Following the open philosophy of Android, there is no reason for Google to ban such applications from the market, since only the installation and use of the application without the device owners authorisation is prohibited. In contrast such applications are not permitted in the App Store due to the restrictions specified by Apple [36].

Teufl et al. [181] highlights that besides from apps which proclaim themselves as spyware, a widge range of *security related software* (e.g. Lookout [133], Norton Mobile Security [59]) exists with the aim to increase the security of the user's phone and data. Such utilities commonly offer features like to locate or remotely wipe a stolen phone, but sometimes even allow to remotely control the device which allows things like to play an alarm sound, record audio, take pictures or read text messages.

Although the main intention of spyware and such security applications is completely different, only the context in which they are used defines if they can be considered as malicious application or harmless software [181]. If spyware or security tools are installed under the user's behalf, both types of applications cannot be judged as malware. However, if such applications are used without the knowledge of the device owner, the behaviour must be classified as malicious, since they actively allow to spy the user without any notice.

2.1.3 Grayware

Smartphones commonly contain a lot of personal information and allow to gain detailed knowledge about the user. *Grayware* spies on users to collect such information, although the companies that distribute grayware do not aim to directly harm the users by this activities [188]. Hence, grayware usually

provides real functionality and value to the user, but often uses the personal information for advertising purposes or user tracking. Reliable companies openly disclose their data collection habits but others use misleading and unclear privacy policies or end-user license agreements (EULAs) to deceive users, therefore is grayware often on the borderline of legality [64].

Developers of free applications often use advertisement to finance the development and monetise their applications. Mobile advertising companies (e.g. AdWhirl [2], JumpTap [127]) provide SDKs which support developers to easily integrate advertisement into applications. To maximise the revenue the advertisement is targeted to the user by supplying additional information like the location, gender, or age. Some advertisers are further interested in identifiers (e.g. IMEI, Android ID; see Section 5.2.3.1 for more details) which allow to uniquely identify a device and track the behaviour of the user over time, accordingly raising privacy concerns for the user.

For the user it is often hard to know in which way the information, an application is permitted to access, is used. The user relies to some extend on the fidelity of the developers, especially that they proclaim for what they use the personal data and how they handle the disclosure of such information. An application might, for example, use the location information to present the position of the user on a map, but at the same time and without notice of the user send the information together with some unique identifiers to an advertising company or remote server.

2.1.4 Motives for Malware Development

Developers have different motives and pursue different objectives for the development of malware. Following some possible motives, which have been seen at malware in the wild to some extent, are highlighted.

2.1.4.1 Novelty and Amusement

Especially in the beginning of the mobile malware development applications have been found which seem to have the intention to *amuse* the author by doing mischief. For example, `Eeki.A` [154] changes the wallpaper of infected iPhones or `Smspacem` [177] sends various short messages with anti-religious content from Android devices. Such malicious applications mainly cause discomfort for the user and can often be seen as *proof of concept* for new attacks and for the demonstration of vulnerabilities.

2.1.4.2 Financial Gain

Research by Zhou [198] shows that *gaining financial benefits* from malicious applications is one of the driving forces for the development of malware. The practices for making money range from direct methods like premium rate calls and short messages to indirect methods where user credentials are misused or sensitive information is sold.

2.1.4.3 Premium Rate Services

One profitable way for attackers is to develop malware which secretly contacts *premium rate services* using *short messages* or by *calling* them in the background. Legitimate premium rate services offer phone numbers for technical support, TV show votings or adult chat lines, or provide short message services for digital content like news alerts, ringtones or wallpapers. A part of the higher fee for the call or short message are paid to the service provider, allowing a payment of the offered service. If attackers set up such premium services for their malicious activities they can directly make money out of it by forcing the malware to contact the service.

Android allows applications to send and intercept incoming short messages in the background without any notice to the user after granting the required permissions at install time (see Section 2.2.1). This

allows malicious applications to contact such services with ease if they got installed once. FakePlayer is the first known malware which uses this methodology [198]. Since then various malware of this kind was developed including more sophisticated ones like GGTracker which use better techniques to hide their behaviour, allow the attackers to remotely change the premium service number and even register users to premium service subscriptions [198]. Geinimi and others have similar functionality but as well allow to call premium numbers in the background [198].

2.1.4.4 User Credentials

Smartphones are nowadays often used for online shopping, banking, email and other services which require *user credentials* including passwords and credit card data. Besides, offer banks two-factor authentication relying on the smartphone as second channel. Moreover, users store even password and payment credentials on their phone using password managers. Accordingly, this kind of usage make mobile platforms a profitable target for theft of user credentials [80] , since gathered credentials like credit card numbers, bank account credentials, email passwords can be sold on the black market or used directly by attackers to make profit [84].

Zitmo and Spitmo are examples for malware previously seen on the Symbian platform and ported to Android [50, 198]. Both applications defeat the second channel of a two-factor banking authentication by intercepting all incoming short messages. Subsequently, the forward the information to a remote server over the network or by using short messages. Besides, the two malicious applications work together with a desktop malware which steals the user's login credentials and deceives the user to install the malicious applications on their smartphone.

2.1.4.5 User Information

Besides user credentials a lot of *user related information* resides on smartphones and is accessible through the Application Programming Interfaces (APIs) of the mobile platforms or might be obtainable by malware through vulnerabilities. Contacts, browsing history, installed applications, the location, personal information (phone number, mail address, postal address, et cetera.) or unique device identifiers to mention only some of the most important ones (A more detailed list can be found in Section 5.2.3.1.). Although it is hard to tell why malicious applications collects such informations, Felt et al. [80] hypothesises that this data is sold by malware distributors for financial gain.

Collected information which allows a detailed profiling of the user might be bought by advertisers or marketing companies to improve product placement and targeted advertisement. Nevertheless, legitimate applications already include advertising libraries which commonly collect personal data, but maybe offer less detailed information about the individuals.

Similarly, detailed information about the device owner or the contacts like the full name, the phone number or the mail address can be sold on the black market [84]. Where such detailed user information is offered and sold for spamming purposes and phishing attacks.

Unique device identifiers like the IMEI have value for the black market of stolen phones [80]. Since, if a phone gets stolen the IMEI is blacklisted by the network providers and thus prevented from connecting to the network. Nevertheless, gathered IMEIs can be used to alter the black listed phones IMEI by a valid one and consequently allow to connect to the network again. However, it is common practice of applications and advertisers to collect IMEIs, meaning the possible large supply of IMEIs lowers the revenue for attackers.

For example, the dubious Android applications known as SndApps uploads information about the user's email accounts, the IMEI as well as the phone number to a remote server in the background [193, 200]. Due to the fact that such behaviour is typical for grayware the applications have been removed from Google Play. Nevertheless, as the G Data Software AG [93] points out, after integrating an EULA

and encrypting the data before transmission, these applications made it on to Google Play again and remain there. This shows another case of misleading EULA and highlights the risk of downloading grayware from the official market.

The stealing of user information or credentials and the usage of premium rate services, as described and affirmed by various actual samples, show common methods used by attackers to gain financial benefits. However, depending on the intention of the attacker financial interests might not only be limited to the motives listed above, even other attacks could be indirectly utilised to acquire money. For example, information gathered by spying or providing resources for spamming activities might as well be sold on the black market.

2.1.4.6 Spying

Use cases for personal spyware in the private area are the *spying* of the partner or the parental control of the children. This allows to find out if your significant other is cheating on you, or to verify that children follow your rules [145].

Companies might use the functionality for business espionage allowing to gain information about the developments of the competitor or to get insight into upcoming business deals, and thus enable to achieve competitive advantage [162]. Another use case for personal spyware is the sneaky monitoring of employees by company executives [145].

Governments could use spyware for monitoring citizens in the large or target it to observe suspected criminals. The threat of the governmental usage might be even more powerful assuming their ability to force network carriers or device manufactures to distribute spyware on all devices [80].

Even though not all of this scenarios have yet been seen in the wild or became known, they show that the privacy of the user is not respected by such spying applications if they are installed and used without the consent of the device owner.

2.1.4.7 Utilise Resources

Research by Zhou [198] shows that a high percentage of malware integrates functionality which allows to remotely control the compromised phone through the network or by short messages using command and control (C&C) servers. This features are used by attackers to create botnets for *utilising* smartphone *resources*. The resources can, for example, be used for spamming or distributed denial-of-service (DDoS) attacks.

Short Message (SMS) and email spam is often used for advertisement or phishing attacks [80]. Since spam messages are illegal in most countries, spammers commonly use botnets to spread the messages. Since the usage of multiple compromised devices obscures the origin of the original message and thus reduces the risk for the spammers to get caught.

Since Android allows to send SMS in the background without any notice, malware could misuse this functionality to send SMS spam. The user might only realise the malicious activities after he receives the monthly bill for all the sent messages, or maybe never notice it if he has an unlimited messaging plan. `Pjapps` [176] is an example for malware offering such functionality. It further allows to force the compromised devices to open a certain web address. Accordingly, if the botnet is large enough it allows to attempt DDoS attacks by simultaneously contact a web address with all hijacked devices.

The utilisation of the resources of a smartphone could cause additional costs for the user for sent messages or high data traffic. Further the device is misused without the knowledge of the user in an unwanted manner and might, moreover, get blacklisted for its behaviour.

2.2 Security Measures

The developers of mobile platforms have taken various security measures to reduce the risk for possible threats. Although the measures vary for the different platforms they can roughly be divided into three steps depending on the time they take place:

First, *preventive* measures should make it difficult to develop malware. Mobile platforms have a security architecture designed with the aim to protect user data, system resources, applications and the device [5]. To achieve this security objectives the platforms choose slightly different approaches, but roughly all provide the following key features [5, 38, 141]:

- The usage of established, robust *security* architectures at the *OS level* (Linux kernel for Android).
- Utilisation of *hardware security features* for data encryption, cryptographic operations and memory management.
- *Sandboxing* of applications to restrict access to system resources and data as well as others applications data.
- Mechanisms for *secure inter-process communication (IPC)* between applications, allowing to safely share data between applications.
- *Application signing* to identify the application author and to build trust-relationships between applications.
- A *permission system* restricting the access to system resources, user data and credentials.

Second, various *detection* methods should help to identify malware that circumvents the preventive measures. The detection of malicious behaviour of applications starts on the distribution channel, it further relies on automated testing of applications, security applications and especially the community to find malware. Such analysis and detection methods are presented in more detail in Chapter 3.

Third, *recovery* methods are required to remove possible threats if they have been detected. Since none of the previous steps can give a total guarantee for the prevention of malware distribution, measures which allow to lower the impact by removing such applications are necessary (more details in Section 2.2.2.1).

This brief description presents the security measures modern mobile platforms are build on. Following, we will describe permissions and distribution channels and their role as security measure in more detail, since both are especially interesting for this work. Permissions have the aim to protect sensitive parts of the devices and restrict the access to certain information, still we want to make the user aware of the remaining risk. Different distribution channels pose different risk of downloading malicious applications and deploy different security measure that we want to highlight.

2.2.1 Permissions

Permissions are used to restrict the access to security and privacy relevant system resources like the network connection, the camera or the GPS and user related data like SMS, contacts or user credentials.

The security architecture of Android heavily relies on the permission system which regulates the access to protected APIs. By default, an Android application is only allowed to access a very basic set of system resources and some of the sensitive capabilities are not made accessible at all by an intentional lack of an API [5]. Prior to the installation of an application the specific permissions required by the application are pointed out to the user. Following a *user-centric* strategy, the user can decide whether to

grant all this permissions for the application or decide to not install the application at all [182]. Accordingly, there exists no possibility to grant just a subset of permissions or to revoke certain permissions after the installation.

The decision, whether to trust an application or not, can essentially just be made by verifying if the required permissions fit into the *context* of the application [189]. The process of deciding whether the required permissions are appropriate, can be challenging. Lets, for example, consider a camera applications for taking pictures. It is obvious that the application has to access the camera hardware. If the application also wants the permissions to access the Internet and read contact data, an experienced user might get suspicious. However, these permissions could just be needed to upload pictures to the Internet, or to directly add pictures to contacts in the user's address book. Nevertheless, a malicious version of the application might as well upload all your contacts to a server in the Internet without your knowledge. So even after granting an application certain permissions, it is hard to track in which way they are used and if private data is secure.

On the iOS platform the user is asked for permission to access restricted resources at the time an installed application first uses the protected feature. Currently, only the use of push notifications, the access to the location information and the Twitter account is protected. However, it has been announced that contacts, calendars, reminders and the photo library will be protected by permission in the upcoming iOS version 6 [126]. The different usage of permissions in the security architecture of Android and iOS reveal the distinct concepts behind them.

The Windows Phone OS follows a similar approach as Android, but uses a coarser categorisation of the permissions. However, for further details about the different deployed permission models of mobile platforms we reference to the paper of Au et al. [42].

Permissions try to limit the risk that applications incorrectly or maliciously access sensitive data. Although this an important security measure the user should be aware of the risk that applications might misuse the access to sensitive data for other purposes like advertising or even malicious activities.

2.2.2 Application Distribution Channels

Mobile platforms provide the user different solutions to download and install third-party applications. The different platforms differ heavily in their security measures and the risk they pose to the user.

2.2.2.1 Official Markets

Apple, Microsoft and Google operate their official markets App Store, Windows Phone Market and Google Play for the distribution of third-party apps [35, 99, 140]. Developers who would like to distribute their applications over the market need to *register for a developer account*. The fee-based registration (US$90 per year for the Apple App Store and 99$ for the Windows Phone Marketplace, US$25 one-time for Google Play) creates a first small barrier for the distribution of malware over the official markets, since a malicious developer must at least acquire a fake credit card for the payment [137].

To publish an application on the market it first has to be digitally *signed by the developer*. The signing of the application allows to identify the author of the application and enables to establish trust relationships between applications. For iOS and Windows Phone the certificate used for signing must be issued by Apple or Microsoft respectively and is related to the developer account whereas Google allows the developer to use self-signed certificates [23, 39, 139]. If applications show malicious behaviour all others applications created by a developer can easily identified by the certificate of the developer and perhaps removed.

Apple and Microsoft follow a contrary philosophy than Google according to the release of applications on their market. Apple rigorously *approves application* prior release on the App Store. Although not all details about the approval process are known, Apple describes that submitted applications are

tested by two reviewers for software bugs, instabilities of the app, privacy violations, exposure of children to inappropriate content and the use of unauthorised protocols [37]. Whereas Microsoft deploys a similar approval process as Apple [49, 139], Google in contrast follows the open philosophy and has no formal review process, but as well conveys in the term's of service that it is not allowed to distribute malware on the market [137].

The markets allow the *community* to *rate* and write *reviews* for applications. Users can decide based on the reputation, if the trust an application. On Google Play users can moreover flag applications as inappropriate what might initialise a inspection by Google. Especially the open market of Android relies on the community as another security layer to identify and report broken or malicious applications [188]. Still, to submit a review a user first has to install the application and might thus already expose sensitive information.

In 2012 Google introduced *Bouncer* a tool providing automated scanning of Google Play for potentially malicious applications [114]. The service automatically analyses new and already published applications for known malware and also looks for misbehaviour of applications that could indicate malicious activities.

Since none of the prevention and detection methods can guarantee totally security, Apple, Microsoft and Google integrated technologies into their platforms which allow to quickly remove malicious application on discovery [143, 148, 164]. With the *remote application removal* feature (or *kill switch*) dangerous applications can be removed rapidly from all devices to prevent further exposure to the user and limit the impact. However, this feature can only ensure its functionality on unmodified systems which are not rooted or customised in any way. Furthermore, malware might block the functionality on a compromised system.

The official markets are certainly the most secure way to find and install applications. Nevertheless, previous incidents show that markets, even with their measures, can not ensure that no malicious application are released [68]. This is especially dangerous since they give the user a sense of security. Hence, it is important to raise the awareness of the risk to download malware even from an official market.

2.2.2.2 Alternative Markets and Direct Download

By default, Android blocks the installation of applications not coming form the official market. The user has to activate an option to allow the installation of non-market applications. *Alternative markets* for Android like the Amazon Appstore [3] or AppBrain [33] offer developers another distribution channel for their applications. They often advertise their markets with higher revenue for the developer or better marketing, search and recommendation features. Although some alternative markets have approval processes and test applications before they get released, research by Zhou et al. [199] shows that the risk of downloading malware his higher for alternative markets.

Besides markets, the user can directly download and install Android application packages (APKs) from platforms like SlideMe [171] or GetJar [97], or any other website. This distribution channel for applications pose a high risk for the user since there is generally no validation and approval through an official authority.

As opposed to this, Apple and Microsoft follow a closed strategy for their mobile platforms and do not allow the usage of other markets or the direct installation of applications. To install third-party applications this devices require a jailbreak (iOS) [119] or unlock (Windows Phone) [161], but doing so does void warranty and poses the risk of damaging the device.

The manual download of applications or the use of alternative markets comes generally with a higher risk of installing malicious applications. Consequently, users need to be aware that attackers might prefer this distribution channels since less effort is required to circumvent security measures on the distribution channel.

2.3 Reasons for Targeting Android

This section investigates reasons why Android got one of the main targets for developers of malicious applications.

2.3.1 Market Share

Potential attackers try to get the best benefit out of their efforts and one way to achieve this is to target the largest audience [128]. Similar to the computer world, there is a relation between market leadership and the interest of attacker to target their attacks on them. Recent market analysis by Gartner Inc. [95] indicate that Android got the largest mobile platform with a market share of 56.1% followed by iOS and Symbian. Furthermore Google Play is the second biggest market (behind the Apple App Store) offering more than 600,000 applications and a total of 20 billion application downloads [125]. This numbers lead to the speculation that targeting malware for the Android platform might be especially interesting for attackers. This is reflected by the report from Juniper Networks [128] which reveals that most of the malware of 2011 is targeted for Android and the numbers are constantly increasing. At the same time there have not been any significant incidents of malware targeting the iOS platform [181] and even less is known about malware for Windows Phone.

Although this significant market share of Android most probably provides the incentive for the development of malware for Android, it can not be seen as the only reason for the dominance. Factors like the market where the phones are sold, the variety of phones in different price ranges, the underlying platform and other aspects might as well play a role.

2.3.2 Openness and Technical Features

Android offers APIs which allow deeper system integration than other platforms like iOS. This open approach, which is often highly regarded by the developer community, brings drawbacks and opportunities. It allows developers to create applications with features not possible on other platforms, but at the same time this capabilities might be misused by malware developers.

A considerable example are background services. The iOS platform limits background activities to certain services provided by the system for things like the location or notifications, in contrast Android allows any process to run in the background [181]. Another example for a feature only available for Android is the sending and intercepting of incoming short messages without any further notice to the user (assuming the user granted the required permissions). This opportunities allow the development of sophisticated application and made the platform attractive for developers. However, not only developers of legitimate applications are tempted by these possibilities.

As described in Section 2.2.2 Google Play has no strict approval of submitted applications like the Apple App Store or the Windows Phone Marketplace. Furthermore, Google is less restrictive in the types of application which can be published on Google Play. For example, applications which openly state that their purpose is spying are allowed on Google Play. Such application are in contrast prohibited in the App Store. The openness and the vague approval of applications on Google Play in comparison to the App Store is another factor which makes it more interesting for attacker to target malware for Android.

Besides the official market, Android allows the installation of applications from other sources like alternative markets or direct downloads by just activating a certain option in the settings menu. Although it might be more difficult to target a that big user base by this distribution channels, attackers have the advantage that there are only limited or no security measures, for validating published applications, integrated by the platform owners. Hence, an Android device might be unconsciously exposed by the user to distribution channels without a legitimate authority approving the published applications.

2.3.3 Version Fragmentation and Heterogeneous Devices

Android is not bound to a certain hardware or manufacturer of a device like iOS with the iPhone. Google only develops the mobile platform and relies on hardware manufactures for the distribution of their OS. Google leaves many options to hardware manufacturers which typically extend the stock Android OS with a customised user interface and additional features. While the possibility for such modifications offer a large flexibility and might give the user more opportunities it arouses some problems. Hardware manufactures which customise their OS need to integrate every operating system update released by Google into their custom variant. This leads to delays in the deployment of security fixes. Further, manufactures tend to not delivery any updates for older hardware caused by the effort needed to integrate the updates. This facts leads to the problem that many Android devices do not receive security updates fast enough or at all and thus can be targeted by malware.

Even if other platforms are currently not so affected by malicious application as Android, there already exist certain proof of concept attacks for jailbreaked iOS [128] devices and also vulnerabilities for Windows Phone [166] have been found. This indicates the general risk for malicious applications, regardless of the deployed platform.

Although various security measures are taken by the platforms different threat types exist and remain. No matter if the motives behind the development of malicious applications are to gain financial benefit, collect personal information or only novelty and amusement, they all pose a certain risk for the user. Hence, it is important to analyse applications to identify their real behaviour. Especially, the Android platform got a main target for the development of malicious applications through facts like the high market share, the openness and the technical features. Accordingly, different application analysis and malware detection methods with varying capabilities and requirements have been developed, which we explain in the next chapter.

Chapter 3

Android App Analysis Methods and Malware Detection

In this chapter we will present Android application analysis and malware detection methods. However, to be able to explain the analysis methods in detail, we first define the underlying analysis environment.

3.1 Analysis Environment

Before we discuss the application analysis and malware detection methods, we have to determine the characteristics of the test environment in which the methods are applicable. The condition of the test system and the possibilities to modify or interact with the system do affect the usage of different analysis methods and limit their capabilities.

If the test environment is already predefined, a suitable analysis method has to be chosen, according to the properties of the environment or must be adapted for the use within it. For example, mobile security applications, used to scan mobile systems for malware similar to antivirus software on desktop computers, have to cope with the properties of the test environment. Hence, limitations like limited resources might lower their malware detection capabilities.

On the contrary, if the analysis environment does not impose any restrictions and allows modifications and advanced interaction, more sophisticated test methods can be applied. Still every analysis method has its advantages and disadvantages: Some allow a detailed application analysis, but rely on the interaction of the user and require a real device, while others offer less detailed analysis capabilities, but only require metadata from the application market and can run autonomously.

Teufl et al. [181] divides the environment properties, crucial for the selection of the analysis method, into five categories as illustrated in Table 3.1 and described below.

3.1.1 Execution Environment

The underlying *execution environment* of application analysis and malware detection methods can be divided into four different types (see first row of Table 3.1): The system of a *standard phone* is not modified in anyway and is in the state as delivered by the device manufacturer. Depending on the manufacturer and the age of the device it might be provided with an up-to-date system version or lag behind with software updates. Additionally, the manufacturer might provide Over-the-air (OTA) updates for the system to upgrade to newer versions or apply security patches. Such an environment is typically used by a *default user* and mobile security applications are commonly deployed on this environment.

A *rooted phone* is modified by the user to attain privileged control and enable additional features of the device. This root access allows to overcome limitations enforced by carriers and hardware manufac-

Execution environment	Standard phone	Rooted phone	Mod. system image	Emulation	None	
System behavior	Superficial state	Detailed state	None			
Available data	Ext. app. metadata	Store metadata	Installed app	App package	Network traffic	Side channels
Interaction	User interaction	System interaction	None			
Application scope	One app	Some apps	Arbitrary apps			

Table 3.1: Categorisation of the environment properties for malware analysis/detection methods in five groups (redrawn from Teufl et al. [181]).

turers. Applications can run with root-level permissions, allowing them to bypass security features like the sandbox, file system permissions and limitations of the API. This more sophisticated environment is frequently used by *advanced users*, and *expert users* use this privileged execution environment for their advanced application analysis methods. Moreover, malicious applications might use security leaks to gain root access on a phone, however, we only consider the intentional rooting of a device at this point. Side note: The similar process of gaining privileged access on the iOS platform is called *jailbreaking*.

Modified system images are customised and extended versions of the mobile system. Arbitrary function can be added to the OS and the provided APIs to extend the functionalities. For the detection of malware, modified systems allow to get a deeper insight into the behaviour of applications by tracking the interaction with the system. Advanced dynamic application analysis methods can be deployed on modified system images.

Emulation of devices, as commonly used by developers during the development of applications, is beneficial for detection of malware. It allows to run automated analysis of applications on powerful machines, emulating a large number of devices with different deployed system images. However, not all aspects of real devices can be emulated and sophisticated malware might detect that it is executed on a emulated environment [149].

Finally, the case (*None*), where no execution environment is available or required for the analysis, has to be considered. An example is the static analysis, where no execution of the application, neither on a real nor on an emulated device, is required.

3.1.2 System Behaviour

The information, available in correspondence to the *behaviour of the system* can be categorised into the following three groups (second row of Table 3.1): The *superficial state* refers to information available and understandable by a default user. The user might recognise unexpected behaviour, like application and system crashes or a increased battery drain (caused by intensive use of background services through malicious applications). Furthermore, the user might notice high bills caused by malware contacting premium services or extensive data transfers over the mobile network.

The *detailed state* indicates that analysis methods have access to additional information such as running processes, the network traffic or inter-process communication (IPC) between applications. The availability and amount of this detailed information is limited by the execution environment as described previously.

None implies that the system behaviour can not be analysed, for example, if there is no execution environment available at all.

3.1.3 Available Data

The available or obtainable information about an application can be classified as follows (third row of Table 3.1): The application markets, offered by the mobile platforms, reveal information (*store metadata*) about the application prior the installation, such as a textual description, the required permissions or the user rating. The market metadata analysis is based on this information. Since the metadata is not included in the application installation package, there is no need for the package file, but if analysis methods want to relate the store metadata with the information from the application package, both data must be acquired separately.

The following three methods require the application package file or the installed version of the application. With access to the *application package* the software can be analysed prior installation. Static analysis is an example for an analysis method which can be applied at this point. For *extended application metadata* additional data, such as information about the IPC-interfaces for the communication with other application or the application entry points are extracted from the package. Again, static analysis uses this extended metadata as well as mobile security applications. An *installed application* denotes an already installed application package on a mobile device or emulator. Dynamic analysis requires the execution of the application and thus relies on the installation of the application. However, an installed application does not necessarily reveal the same information as available from the package file, since some data might be omitted during installation, but on the other hand the execution reveals other important information.

Information, which is not directly available for analysis methods, can sometimes be obtained through *side channels*. For example, the evaluation of system log files could reveal important information about the actions of applications.

The *network traffic* refers to the incoming and outgoing data on the network interfaces caused by the installed applications or the OS of the mobile device. The network traffic might, depending on the limitations of the execution environment, be directly captured on the mobile device or intercepted externally (more details in Section 5.1). The analysis of the network traffic might reveal crucial information about the external communication of applications and the transfer of sensitive data. The network traffic is the main focus of the analysis method, developed and presented in this work.

3.1.4 Interaction

Each analysis method has different requirements on the level of interaction with the application during a test (see fourth row in Table 3.1): Methods like the static analysis require no interaction (*None*) of the user or the test environment with the application and can run autonomously. Whereas methods, such as the dynamic analysis or the network traffic analysis, require *system* or *user interaction*. Automated analysis interacts with the system by means of starting and instrumenting applications and simulating user input. Certain analysis methods even require the time-consuming interaction with the application through a real user.

However, generally the more complex the interaction with the system is, the more information can be gathered by the analysis method. At the same time, the high level of interaction requires sophisticated test systems or the interaction of humans, as a result this methods are classified as time-consuming and expensive processes.

3.1.5 Application Scope

As last property of the analysis environment we consider the number of different applications, which can be covered by the different analysis methods (last row in Table 3.1): Certain analysis methods are only able to check *one application* at a time and can not give information about potential interaction with other

applications. Moreover, although methods like the network traffic analysis can capture and inspect the traffic of multiple applications, they can not associate the gathered information to a certain application.

Other methods allow to analyse *some applications* in the same scope at the same time, but are limited to a certain number of applications due to the technical capabilities of the analysis method or by the number of available applications. A typical example for a method, applied in such an environment, are mobile security applications, which can only examine the applications present on the device.

The access to an *arbitrary* number of *applications*, as possible by application store providers, allows to apply analysis methods on a multitude of various combinations of applications. The analysis of several applications in the same scope can help to identify malware, that distributes its malicious behaviour over multiple applications or exploit capability leaks caused by unprotected IPC-interfaces or weak system applications [53, 109, 153].

All the denoted properties of the analysis environment heavily influence the selection of analysis methods and impacts their capabilities. The overview of the properties should help to identify important characteristics of the contemplated system and assist in the selection of applicable analysis methods, presented in the next sections.

3.2 Analysis and Detection Methods

In the following, we describe application analysis and malware detection methods in greater detail.

3.2.1 The User

Although *the user* can not be considered as analysis method, he plays an important role in the identification of malware. A user might get suspicious prior the installation of an application, if the required permissions are doubtful in the context of the application, or he might identify unwanted and malicious behaviour of already installed applications. Furthermore, security experts play a key-rule in the application of in-depth analysis methods, thus the user can intervene at various stages to detect malicious applications. The capabilities to recognise unwanted or malicious activities heavily depends on the knowledge of the user about the mobile platform and existing threats for it, hence we divide into *default-*, *advanced-* and *expert-users*.

3.2.1.1 Default User

We consider that a *default user* is generally only capable to use a mobile device and its applications in the intended manner without considering the related security risks. The default user has generally no in-depth knowledge about the permission and their ramifications as well as he does not know about the aggravated risk of downloading applications from alternative markets. Default users are easily tempted to install malicious applications, which use misleading descriptions or masquerade as popular applications. Moreover, such user's are only able to recognise very obvious malicious behaviour and most of the time only in the aftermath.

The default user can be classified as follows, relating to the properties of the analysis environment we discussed in the Section 3.1.1: The default user works on a *standard phone* without any modifications and can only observe the *superficial state* of the system, including information like crashing applications or high bills. Since the *user* directly *interacts* with the application, he might cause the execution of the malicious functionality. Nevertheless, the user might eventually be able to identify the interaction between *some applications* on a very superficial level, such as if one application ask for the credentials of the other one. The default user generally only works with the *installed application* and does not bother to work with application packages. The most helpful information available for a default user is the *store metadata* including data like the description of the application, the user rating, comments or the required

permissions. If an application has a low rating and comments include complaints, even a default user might be convinced to not install such an application. On the other hand, if only the information from the description and the required permissions are available, a default user might be overwhelmed with the task of deciding weather an application is trustworthy.

Besides user with basic understanding, we can classify users with advanced knowledge, described in the next section.

3.2.1.2 Advanced User

Advanced users have more knowledge about the mobile platform and know about general threats arising from alternative markets and third-party applications. The advanced knowledge allows the user to better understand the required permissions of applications and to judge upon the danger of approving certain permissions. Advanced users further know how to install customised and modified system versions, which allow to access additional features. Moreover, mobile security application and other security tools are typically deployed by user's with this proficiency. Although, the greater knowledge lowers the risk of installing malicious applications and allows to identify malware, the chance of being exposed to malware remains.

Advanced users are related the following properties of the analysis environment: Depending on the know-how, the user might use a *rooted phone* or even a *modified system image* to enable desired features and to deploy additional security features. Such security related modifications of the system include functionality like the fine-grained permission management at the run-time [46, 115, 147, 195]. Furthermore, the use of additional tools such as mobile security applications or process monitors allows the advanced user to obtain a more *detailed state* of the device. Since advanced users download applications from various sources, they get in touch with *application packages*, but do not have the capabilities to analyse them. Especially the *market metadata* can be interpreted better by advanced users. Finally, the application scope and the user interaction remains the same as for a default user.

Alongside, advanced users, there are expert users with even more profound knowledge, described in the following.

3.2.1.3 Expert User

Expert users have a profound knowledge about the mobile platform and develop or work with tools for the analysis of applications and the detection of malware. With the in-depth knowledge their possibilities are primarily limited by the methods used for the analysis and the available resources. Thus, the properties for the analysis environment are hard to define at that point and we reference to the analysis method applied by the expert. The research by experts on different security threats and the development of counter-measures are very important to lower the risk, for default and advanced users, of getting exposed to malicious applications.

The user will always play a key role in the identification of malicious applications. Just like malware is developed by users, only users are able to develop detection methods and defence mechanisms. Especially the decision, at which point the behaviour of an application is considered as unwanted or even malicious can only be done by a user in a first step. Expert users fine-tune their analysis methods with their knowledge to provide optimal detection functionality and best possible protection.

3.2.2 Mobile Security Apps

Mobile security applications, like Lookout [133], Norton Mobile Security [59] or Kaspersky Mobile Security [130], offer similar functionality, known from anti-virus software deployed on desktop computers. The anti-virus part of such applications mainly uses signature-based malware detection. *Signature-based*

detection [116, 190] searches for previously defined characteristic (signatures), unique for certain malware, during the detection. The process of creating signatures, that represent the malicious behavior exhibited by applications, is done by human experts. This defined signatures are all stored in a central repository, where they can be retrieved by the security applications. A major drawback of this methodology is, that depending on its capabilities, it can only detect known malware with a signature stored in the repository, and thus might be vulnerable for zero-day attacks.

The improved performance of modern devices allow to deploy heuristic analysis to overcome this drawback and provide better detection capabilities. In contrast to signature-based detection, heuristic analysis tries to identify common functionality and features of malware [190]. This method should allow to even recognise new malware based on the features, known from previous malware.

Besides this anti-virus capabilities mobile security application commonly offer additional features like anti-theft and missing device services (e.g. locate device, remote-wipe) or number based call and SMS filtering. However, the capabilities vary for different mobile platforms, according to their functionalities and restrictions.

Concerning our analysis environment in Table 3.1 following properties hold for mobile security applications: They can run a *standard phone*, but might offer even more functionality, if deployed on a *rooted phone*. They have knowledge about the *superficial state*, but might even use the *detail state* to observe running processes. They run their test on *installed applications* and are as well able to test *application packages*. Mobile security applications automatically scan all installed applications (*some applications*) and require in general no user interaction (*None*), only if they detect certain threats they might ask for user input.

Mobile security applications are important for the detection of known malware and can even be used by novice users. However, they often fail in the detection of new malware and research is required to identify new threats and keep the repository up-to-date.

3.2.3 Static Analysis

During the *static analysis* the source code of an application or the corresponding installation package is examined, to inspect the behaviour of the application and detect possible malicious actions. The thereby applied methods vary in their functionality and capabilities, depending on the available resources: If only the application installation package is available, just general methods like signature-based malware detection, as commonly used by mobile security applications, can be applied. However, to deploy more advanced methods, reverse engineering tools are required, which extract data from the binary installation packages, such as the Android manifest file, the compiled class files and other resources.

The Android manifest is a binary Extensible Markup Language (XML) file [6] that contains information, required by the system to run the application. The manifest describes the components of the application like activities and services. Furthermore, the permissions, required by the application to access protected parts of the API, are declared. Additionally, it is possible to define which permissions other applications are required to have in order to interact with this application. The binary XML is convertible to human-readable XML with special tools such as *AXMLPrinter* [169], *android-apktool* [165] or *Androguard* [32].

The class files, included in the application package, are provided in the Dalvik Executable (DEX) format to be directly executable on Dalvik, a process virtual machine optimised for mobile platforms and used by the Android OS. Since Android applications are commonly written in Java and compiled to Java bytecode (only executable in the Java Virtual Machine), they have to be further converted to Dalvik-compatible executables. To apply static analysis on the data of this executables, they have to be reverse engineered and decompiled to reconstruct the former source code of the application or to at least extract abstract information about the program functionality.

Special tools such as *smali/baksmali* [44], *dex2jar* [192], *ded* [77], *android-apktool* [165], *Andro-*

guard [32], and *APKInspector* [57] together with already established Java decompiling software like *JAD* [158] or *JD* [75] accomplish this task, but with certain limitations. This limitations arise from code obfuscating techniques, which complicate or prevent the reverse engineering process [29, 167]. Such code obfuscation methods are commonly applied by developers to protect their applications and the corresponding implementation from being copied by others. However, obfuscation methods are even applied by attackers to hide malicious code and hamper the detection of malware [40, 150, 194].

3.2.3.1 Techniques

The static analysis of the original or reconstructed source code of an application allows to identify various aspects of an application by applying different analysis techniques without requiring the execution of the program [77, 168]:

The *control flow analysis* enables to construct a control flow graph (CFG), representing all possible execution paths of an application. The specification of certain unwanted execution sequences allows to detect undesired behaviour of applications (e.g. if an intent contains sensitive data and is sent unprotected without specifying the target component).

The *data flow analysis* allows to calculate the possible set of values at a certain point of the program and its propagation according to the CFG. The definition of rules, regulating the flow of sensitive information from one point in the application to another (e.g. from reading the phone number and using it at a point where data is sent over the network), helps to identify suspicious behaviour of applications.

The *structural analysis* allows for declarative pattern matching on the abstraction of the source code without taking the execution path or data flow into account. For example, a pattern for the call of a certain function in a specific class could be defined (e.g. calling `getDeviceId()` in `TelephonyManager`).

The *semantic analysis* allows to specify a limited set of constraints for the source code of the application. For example, a constraint could be that a parameter of a method is not a constant (e.g. to proof that the phone number used in `sendTextMessage()` is not hard-coded).

Besides this more general static code analysis methods, Android specific methods can verify the use of the properties, defined in the Android manifest. Especially the use of the permission and the related call of the system API is of great interest.

The static analysis can be categorised, according to our analysis environment (in Table 3.1), as follows: No execution environment (*None*) is required, since the application is not executed during static analysis. Accordingly, the system behaviour can not be analysed (*None*) and no interaction (*None*) of the user with the application is required. The *application package* or even the source code is required to apply static analysis. *Extend application metadata* can be extracted from the application package and some analysis tools might even take the *store metadata* into account. The scope of the analysis ranges from *one application* and might extend to *some applications*, depending on the application and the definition of interfaces for the communication with other applications.

3.2.3.2 Tools

A variety of static analysis tools for Android exist, applying different techniques and observing different aspects during their analysis:

Stowaway [82] determines the set of API calls of an application and maps them to the according permissions. Afterwards, the used permissions are compared with the predefined permission to identify overprivileged applications.

Kirin [78] offers a advanced application installer, which extracts a security configuration from the Android manifest of an application and matches it against a collection of defined security rules.

ComDroid [53] examines the application interaction and identifies potential component and intent vulnerabilities by applying flow analysis and inspecting the Android manifest.

SCanDroid [91] offers automated security certification of applications by analysing the data flow through and across components.

woodpecker [109] aims to identify leaked permissions or capabilities, using a combination of data- and control-flow analysis.

DroidMOSS [197] extracts certain features of the application code and applies fuzzy hashing (as fingerprinting) to identify modification and repackaging of applications.

DroidRanger [199] combines static and dynamic analysis, where the static part uses a permission-based filtering, behavioural footprint matching and heuristics-based filtering to identify known and unknown malware.

AASandbox [47] uses sandboxing to have an isolated and observable test environment and deploys static and dynamic analysis. The static analysis part scans the code for suspicious patterns and clusters the extracted patterns to identify relevant indicators for suspicious behaviour.

Androguard [32] is a toolkit for the analysis of applications offering a broad variety of functionality, such as reverse engineering capabilities, diffing of applications to identify repackaging, risk indication of possible malicious applications and the possibility to develop new static analysis on top of it.

dexter [134] is an upcoming project with the aim to provide a platform for the collaborative analysis of android applications. Furthermore, automated control- and data-flow analysis, automatic semantic based tagging and code classification should allow to gather a comprehensive set of information about an application.

The static analysis allows to quickly test applications without requiring a execution. Different analysis techniques allow to get an insight about the behaviour of applications and to identify possible malicious actions. However, every analysis relies on the availability or reconstruction of the original code or an abstraction of it. Furthermore, certain malicious behaviour might, through obfuscating or other measures, not be identifiable with static analysis. As a consequence some tools try to use the knowledge, gathered in the static analysis, to instruct the dynamic analysis or combine the result of both methods.

3.2.4 Dynamic Analysis

During the dynamic analysis the application is executed on a real or an emulated device and the behaviour of the application is monitored. The execution of an application is quite time consuming and requires certain resources, to overcome this drawback the use of emulated devices allows to simultaneously execute different tests on a large number of virtual devices. However, a virtual device can never emulate all aspects of a real device.

An advantage over other analysis methods, like static analysis, is the access to all the available data on the device on the possibility to analyse the detailed state of the device. Profound knowledge about the data, like unique identifiers, the phone number or the stored contacts, opens up further opportunities to track the flow of the information throughout the system. Furthermore, exact runtime information, like pointer addresses or parameter values at a certain point of the program execution, allows to better understand and analyse the behaviour of applications.

3.2.4.1 Techniques

To gather additional information or change the behaviour of the application and the execution environment, *code instrumentation* or an *instrumented environment* is used. *Instrumentation code* is used to intervene at a certain point of the program, whether to log the information at this execution stage or to modify some data to change the behaviour of the application. The addition of such instrumentation code can however be quite difficult, if only the installation package is available, and might cause side effects resulting in unpredictable behaviour.

The *instrumentation* of the *environment* requires extension or modification of the original system. Thus, this method might not be applicable in every case, since it presumes to use of a modified system image or the root access to modify certain parts of the system. However, it allows to augment analysis code or to modify parts of the system like the API. The latter is commonly used by tools to log the access to privacy sensitive parts of the API or to return empty or modified data.

To provide a comprehensive test of an application, most applications require the interaction of the user to cover all execution paths of the program and to activate every functionality. However, the manual user interaction is very time consuming, thus many dynamic analysis methods try to simulate the user input. The user input simulation reaches from code instrumentation, based on the information from static analysis to random generated input, known as monkey testing. Since only executed paths of the applications will be considered during the analysis, it is important to cover all execution paths, since otherwise malicious behaviour might be undiscovered.

Considering our analysis environment, described in Table 3.1, the properties of dynamic analysis are as follows: The requirements on the execution environment vary heavily, depending on the specific deployed analysis method. Methods, where code instrumentation on the application package is applied, can use real devices, no matter if they are *rooted* or *standard phones*. Others, which for example use instrumented environments, might utilise *emulated* devices to run automated tests. Independently, *modified system images*, including instrumented test environments, might be deployed on emulated or real devices. The access to *all* available data during dynamic analysis allows to gain information about the *detailed state* of a phone. The *interaction*, required for the dynamic analysis, might be done by the *user*, but for automated testing simulated input or the *interaction* with the *system* is needed. An advantage of dynamic analysis is the possibility to analyse *some applications* at the same time and thus to identify communication between them.

3.2.4.2 Tools

Following we list some dynamic analysis tools with different aims and capabilities:

Apex [147] modifies the permission framework of Android to allow for selectively granting and revoking permissions at runtime and imposing constraints on the usage of resources.

AppFence [115] extends the Android OS with two additional privacy controls: **First**, it substitutes data, that the user wants to keep private with fake data. **Second**, the transmission of data that is permitted to be used by an application on the device only, is blocked from being transferred over the network.

MockDroid [46] is a modified version of the Android OS, which allows the user to mock an application's access to a resource. At this approach the access to a certain resource can be revoked during the run-time by reporting to the application, that the resource is empty or not available.

TaintDroid [76] is an extension to the Android platform, that tracks the flow of sensitive data through and out off the system by automatically applying labels (taints) to the data from privacy-sensitive sources.

DroidBox [155] aims to offer a sandbox for the automated testing of applications, using TaintDroid and to supply a tool, which filters relevant data and provides a rich representation of the results. Besides various other results, a list of the contacted servers is presented.

TaintDroid Runner [65] is as well an extension of TaintDroid for the automated analysis of applications in an emulated environment with simulated input. Furthermore, the tool should indicate leakage of privacy sensitive information and the generation of costs for the user.

DyAna [131] uses the static instrumentation approach to inject logging code into the application installation package. The log generated during the run of the application on a device or a emulated system can be analysed with the framework to present a static structure tree, call graphs, heuristics and histograms.

DroidRanger [199] as mentioned before, DroidRanger has a dynamic and static analysis part. The dynamic analysis part uses the information about untrusted code, gathered by the heuristic from the static analysis, to deploy dynamic execution monitoring to inspect the runtime behaviour of this suspicious parts.

AASandbox [47] executes the application in a sandbox, which intervenes and logs low-level interactions with the system during the dynamic analysis. The log file is summarised and reduced to the important information for better analysis.

Bouncer [114] is a tool, developed by Google, which provides automated analysis of applications submitted to Google Play. Although not all technical details are publicly available, Google announced that all applications are tested on emulated devices in their cloud infrastructure for hidden malicious behaviour and are compared to known malware. A major advantage of Bouncer is the access to a broad range of additional information, including meta data from the store, other applications and information about the developer account.

A drawback of dynamic analysis is that malicious application might detect that they run on a emulated device or a modified system and prevent the execution of their malicious code. Furthermore, attackers might identify the characteristics of the test environment and adopt their malicious behaviour to circumvent the detection mechanisms. Oberheide and Miller [149], for example, presented methods for the dissection of Bouncer and O'Kane et al. [150] explains methods for obfuscating malicious behaviour.

Although dynamic analysis is quite expensive, it might present the most realistic environment for an application and especially for malware. The real execution of the application and the analysis of the information monitored during the execution might provide solid information to identify possible malicious behaviour.

3.2.5 Market Metadata Analysis

The market metadata analysis focuses on the aspect, that most of the malware relies on the user to get installed. Since the user decides, mainly based on the metadata information available on the market, whether to install an application or not, metadata analysis aims to analyse exactly this information.

The metadata, available on Google Play, includes details about the application such as a textual description, the category, the required permissions, the user rating and the developer. Metadata analysis should help to get a better understanding about the offered applications, the used permissions, the different application types, the application developers and especially the relation between the application and the required permissions. Furthermore, it should help to get an idea about what is an anomaly or suspicious constellation of this information belonging to a malicious application.

3.2.5.1 Techniques

As a first step the metadata, provided on the market, must be gathered. To identify interesting correlations between the information it is important to have a big information base. However, the huge amount of data requires the use of sophisticated methods to gain knowledge from the metadata.

Simple *statistic approaches* allow to easily gain first information, but can only focus on very specific aspects. An example would be: How many applications require three or more privacy related permissions [189]?

To furthermore extract yet unknown relations more sophisticated *knowledge discovery techniques* are required. Teufl et al. [181] shows an approach, using *machine learning* in combination with a new semantic-aware transformation to identify *semantic patterns*. The extracted semantic patterns build the base for the further analysis and allow to identify interesting correlations between properties of the metadata.

The following properties of the analysis environment apply to the market metadata analysis (see Table 3.1): Since the application is not executed and no system is observed there is no execution environment (*None*), no knowledge about the system behaviour (*None*) and no user or system interaction (*None*). The data, available for the analysis is, limited to the *store metadata* and the application scope ranges from *one application* up to *arbitrary applications*.

3.2.5.2 Reports/Tools

Vennon and Stroop [189] presents in "Android Market: Threat Analysis of the Android Market" a simple statistical analysis of the permission usage of Android applications based on the market metadata.

Teufl et al. [181] presents in the paper "Malware Detection by Applying Knowledge Discovery Processes to Application Metadata on the Android Market (Google Play)" statistical approaches for the analysis of the market metadata as well as the application of machine learning in combination with a new Semantic Pattern Transformation as sophisticated knowledge discovery technique for the store meteadata.

The market metadata analysis provides a high-level method to quickly identify patterns in the metadata description, leading to traces for malicious applications. However, without further in-depth analysis the results might be quite general and include many false alarms. Still, metadata analysis plays an important role to identify and limit the number of possible candidates for more detailed analysis methods.

3.2.6 Network Traffic Analysis

The network traffic analysis is actually a subfield of the dynamic analysis. Since the network traffic is captured or immediately analysed during the execution of an application we still highlight this analysis method separately due to the importance for this work. Besides SMSs and calls, the network is the main communication channel of mobile platforms and their applications with the outerworld. No matter if the device is connected over the mobile- (3G, GPRS) or the wireless-network (Wi-Fi), data is commonly transmitted by applications to servers on the Internet. To study the behaviour of applications, the network traffic can reveal important information about, which servers are contacted, what communication protocols are used or especially how much and which (possible sensitive) information is transmitted by the applications.

The analysis of network traffic in general is already well researched and comprehensive analysis tools like *Wireshark* [191] exist. However, to target the analysis on the special aspects of mobile platforms like the leakage of private information by malicious applications, the remote control of malware by

C&C servers or the misuse of devices for DDoS attacks, the adoption to the underlying environment is required. A detailed knowledge about the characteristics of the environment is essential for the in-depth analysis of the network traffic. For example, only if you know about the unique identifiers of a mobile platform, you can look for them in the network traffic. Less research and tools can be found for this special task, thus this thesis will focus on this area and provide a tool for the network traffic analysis.

3.2.6.1 Techniques

Most of the currently deployed techniques for the analysis of the network traffic pursue the following concept: **First**, the *traffic* is *captured*, whether directly on the device using a cross-compiled version of *tcpdump* [179] and on new versions of Android by using the VPN-service [54], or externally by connecting the device to a special Wi-Fi setup for capturing network traffic. A real device can be used as well as an emulated device during capturing, where the emulator of the Android platform even offers the possibility to directly capture the traffic[1]. The advantage of the external capturing is that man-in-the-middle attacks can additionally be used to decrypt secured network traffic. However, for well implemented applications this method might not work and such a test-setup complicates (especially automated) testing.

Second, the network traffic, captured in the first step, can be analysed, regarding various aspects. A *general/statistical analysis* might give answers to questions, such as: How much data is transmitted? Which servers are contacted? Where are the servers located? What protocols are used? Furthermore, a *in-depth analysis* of the transmitted data reveals, what data is actually transmitted. To apply a more detailed analysis, knowledge about the tested environment, the device and its state are required.

Although, established tools like *Wireshark* offer a broad functionality for this kind of analysis, they must be seen as tool for manual analysis. There is no functionality available, to directly highlight the exposure of private information, leaked from the test device. Testing must be done thoroughly and requires profound knowledge.

During the analysis of the network traffic it is, like for the dynamic analysis, important to reach all parts of the application during the execution, to provide a comprehensive test result. As a consequence, the manual interaction of the user with the application or methods for simulated user input are required.

Related to our analysis environment in Table 3.1 the following properties hold for the network traffic analysis: A *standard phone* is sufficient for external capturing or new VPN-service based methods, whereas the use of *tcpdump* requires a *rooted phone*. *Emulation* of devices is especially useful for the automated analysis of the network traffic. Although not required for general network traffic analysis, knowledge about the *detail state* of the system might help during the in-depth analysis. The available data is limited to the *network traffic*, but *user interaction* or *system interaction* is essential for a comprehensive testing. The scope is limited to *one application*, since it is impossible to distinguish based on the network traffic, between multiple applications.

3.2.6.2 Reports

Following we present some research found for the analysis of the network traffic from Android applications. Since, not that much research can be found yet, we even briefly present some related tests, done for the iOs platform:

The Wall Street Journal [183] in cooperation with the technology consultant David Campbell, analysed the network traffic of 101 popular iOS and Android applications for their article “What They Know - Mobile”. The analysis covered the transmission of private data like the username, password, contacts, age, gender, location, unique device identifies and the phone number to servers

[1] Example usage to directly capture the network traffic at the emulator: `emulator -tcpdump <output-file>.cap`

of the application developer and other third parties. The report shows that applications share this personal data widely and regularly, especially with advertising companies. As a result of the investigative series the Wall Street Journal released an interactive browser for the acquired data, which visualises the transmission of sensitive data to the different servers. Furthermore, the journal asked the application developers, in which way the transmitted information is used and presents this information.

Unfortunately, not all details about the analysis and its methodology are released. The following is known: The traffic was captured externally from the devices on the Wi-Fi, hence, presenting an isolated network environment. Only one application was tested at a time, and to simulate normal use, every application was used for five minutes by a human. Encrypted-traffic was decrypted, using the tool *mallory* [118]. Details, about how the private information was detected in the network traffic, were not revealed.

Fulton [92] held the talk "Cellular Privacy: A Forensic Analysis of Android Network Traffic" at the *DEF CON 19*, where he presented his forensic analysis of the network traffic from Android applications. In his investigation he analysed the transmission of private user information to application authors and third parties. Furthermore, he presented a list of the most contacted servers and his findings on the leakage of sensitive information by various applications. Further, he hypotheses that sensitive data is transmitted mainly for advertising purposes.

The 10 selected applications were tested manually similar to the approach from the Wall Street Journal. The test device was connected to a Wi-Fi where he captured the network traffic and used SSL Strip [146] to decrypt secured traffic. For the manual analysis of the captures he used Wireshark [191] to get a first insight and identified leakage of information, using simple string matching tools.

Smith [172] presented in his paper "An Analysis of Application Transmission of iPhone UDIDs" the use of the unique device identifiers on the iOS platform, following a similar approach as Fulton. The analysis of the unencrypted network traffic reveals the common use of such identifiers and Smith points out the risk to unintentionally correlate this unique identifiers with personally-identifiable information.

Cortesi [61] extended in "How UDIDs are used: a survey" the research from Smith to even analyse encrypted network traffic, using *mitmproxy* [62] to decrypt the traffic. The survey revealed, as expected, that applications even transmit such identifiers over secure channels.

The analysis of the network traffic is a simple, but efficient method for the identification of exposure of sensitive information and reveals a lot about the behaviour of an application. Not only might the analysis help to detect malware, but especially uncover the use of sensitive data for advertising purposes. Currently released research only tested a limited number of applications and used tools from general network traffic analysis. The development of tools, specialised for the analysis of the network traffic from Android application, could help to easily identify data exposure and present the results in a decent way. Furthermore, automated network traffic analysis could allow to test a large number of applications.

3.2.7 Summary

This chapter presented a wide range of existing application analysis and malware detection methods for Android applications. Each method focus on different objectives during the analysis and is applicable in different analysis environments. Some applications require the expensive interaction of the user, however they reveal comprehensive information about the application and its behaviour. Whereas others run completely automated, but deliver less detailed information. Nevertheless, none of the methods is far

superior to any other. Thus, the user has to decide, based on his desired result, the analysis environment and the available time and resources, which analysis methods suites his requirements the best.

Since every method has its advantages and disadvantages a combination of different analysis methods is another approach to systematically identify unwanted and malicious behaviour. High-level methods, which require no user interaction, like market metadata analysis, allow to narrow down the number of applications interesting for more detailed analysis. A user assisted in-depth analysis, with methods like network traffic analysis, could verify in a next step the unwanted behaviour of the circumscribed applications.

Although, there was already quite some research in the field of application analysis and malware detection, there is still space for improvements. Especially, the research by The Wall Street Journal and Fulton [92] demonstrated the potential of the network traffic analysis. However, their research results are limited to a small set of applications and no tool, which could assist in the analysis process was developed.

Chapter 4

Potential of Specialised Network Traffic Analysis for Android Applications

The research, presented in Section 3.2.6.2, demonstrates the heavy use of sensitive user data for advertising purposes by Android applications and shows the potential of the network traffic analysis to identify the leakage of information over the network. Furthermore, the general inspection of the network traffic allows to gather various information about the communication of applications and the thereby used technologies. Therefore, the network traffic analysis plays an important role in the application analysis process and the identification of malicious behaviour.

Researchers currently utilise generally known tools (e.g. Wireshark [191]) for the network traffic analysis of Android applications. Such tools are not specialised for this specific use case and must be operated from expert users with detailed knowledge about the platform, the available data and the possible transmission channels for thorough testing. Without any automation the manual analysis limits the number of applications that can be tested. Consequently, the manual network traffic analysis is costly in terms of time and the results heavily depend on the analysis methods deployed by the expert. Additionally, with various applied tools it might even be hard for an expert to determine all important characteristics and recognise certain correlations.

Following we will first highlight some general aspects about the network communication of an application, which can be enlightened with network traffic analysis. Subsequently, we describe the potential of the in-depth analysis to reveal unwanted behaviour, especially the leakage of sensitive information. Finally, we point out, which of the threat types arising from Android applications can be identified with network traffic analysis. For all this topics we emphasise the importance of a tool integrating all the described methods and undergird how a specialised tool could assist the experts in the analysis of the network traffic of Android applications.

4.1 General Network Traffic Analysis

Network traffic analysis in general is capable to reveal basic information about the external communication of an application. This information can be used to understand and reconstruct the internal behaviour of the application and enables to get a first impression of an application. Following we will highlight some of the general obtainable information, especially interesting in the application analysis.

How much data is transmitted? The information if and how much data is transmitted is a first indicator during the network traffic analysis. It reveals the unwanted extensive use of the network, resulting in charges for the user by exceeding the data plan. Furthermore, it is a first hint for the undesired transmission of sensitive data. A fictional example would be that a camera application, claiming

that it requires the Internet permission for advertising purposes only, sends private data like pictures over the network to a dubious server.

Although, the information in general is quite vague, since certain applications (e.g. YouTube, Skype) require the extensive use of the network, and new Android versions (since 4.0) already have a tool included to monitor the traffic amount produced by every application. This information should anyway be considered in the overall analysis process.

What kind of traffic is produced? The information about the protocols used for transmission uncovers the underlying technologies, used by the applications. Particularly, it is of interest, if applications with access to sensitive information use secure protocols for the transmission of the data. Moreover, the use of inappropriate protocols for certain applications could reveal malicious behaviour. For example the use of the Simple Mail Transfer Protocol (SMTP) to transmit private data via mail.

Where is data transmitted to? Information about the contacted servers helps to track the flow of data and allows to identify communication patterns. A statistical representation of this data could help to quickly interpret the information. In addition, the contacted servers could be matched against a database of servers known for advertising or malicious activities to characterise their behaviour.

In the same manner information about the geographical location of the contacted servers could help to identify countries with a higher potential risk of malicious activities.

At which point of the execution? The exact information about, at which point of the program execution how much data was sent, can be utilised to narrow down the moment at which an application shows its malicious behaviour. Similarly, it could put on display that a application sends data over the network in the background without the interaction and knowledge of the user.

This summary of selected points, which can be taken into account during the general analysis of the network traffic, should highlight the potential of the method. Although, a variety of already established tools for the analysis of this general aspects exist, there was till now no tool available, which automatically aggregates all this information and is well adapted for the special use case of Android application network traffic analysis. A tool providing a rich presentation of the information with additional features like filters for certain information and the possibility to compare the result with previous tests and other applications, is able to provide a solid platform for the analysis of applications and detection of malware.

Even if the general analysis allows to identify first signs of malicious behaviour, the detection capabilities are quite limited. The in-depth analysis of the transmitted data, presented in the next section, is however able to reveal more details.

4.2 In-Depth Network Traffic Analysis

The network is the main interface to the outerworld of a mobile device and almost all data leaving the device has to pass through this point. This fact reinforces the great possibility to identify leakage of sensitive information with the network traffic analysis. However, to detect sensitive information in the network traffic, thorough inspection methods and comprehensive knowledge about the used communication technologies and the available data on the device are required.

Communication Protocols Different technologies and protocols, used for the transmission of data, require individual methods to inspect the transmitted data for certain information. Some protocols transmit the data in simple plain-text format easy to scan, others however use specialised formats, obfuscation or even encryption for the payload and hamper the analysis. Nevertheless, a analysis tool implementing automated inspection methods for crucial protocols is already capable to identify leakage in the most important data transmission channels.

Available Information To inspect the network traffic for leakage of sensitive data, one has to first know which information resides on the device that could be exposed. The determination of this analysis parameters requires profound knowledge about the platform and is a work-intensive task. A tool aggregating established and important analysis parameters helps to automatically analyse the network traffic for this specific aspects and facilitate the work of expert users. Experts would only need to adjust the prescribed analysis parameters for the currently tested system and could focus on the results and the identification of new analysis parameters.

A comprehensive list of the available information on a common device running the Android platform will be given in Section 5.2.3.1.

Inspection Methods The knowledge about the exact information present on the device can be utilised by an inspection method to *directly match* the available data against the transmitted data to identify information leakage. However, the data might be modified, formatted differently or even obfuscated before transmission. For example, the location information could be formatted in different geographical units or only the hash value of a unique device identifier could be transmitted. To still be able to identify such modified information during the analysis, the inspection methods have to be adapted with the knowledge of experts to cover as many modified versions of the data as possible. For instance, the transmitted data should not only be scanned for the exact unique device identifier, but also for common hash values of it.

Other inspection methods could use *regular expressions* to match data of a certain structure to identify information leakage. Common expressions could be defined for very specific data like mail addresses, phone numbers or even for certain identifiers. Nevertheless, both inspection methods cannot cover all variations of data modification or define expressions for all kind of information. Consequently, not every data exposure might be revealed by the inspection methods and incorrect matches could lead to false alarms. Still, a automated analysis tool with different implemented inspection methods is able to present all found records to the user and leaves the final decision about the correctness and importance of the gathered information to the judgement of the expert.

In-depth network traffic analysis, utilising the information about the available data on the device during the analysis, is able to simply but still efficiently inspect the traffic for exposure of sensitive information. However, the manual identification of the available and relevant information and the inspection of this aspects with different methods is time-consuming and requires profound knowledge. A tool, combining established inspections methods in an automated analysis process and providing a rich representation of the results, could help experts in the fast analysis of applications. Automation allows to test a large number of applications and experts could focus on the improvement and detection of new analysis aspects and the identification of correlations in the results.

4.3 Which Threat Types Can Be Detected

Following we describe, which threat types for mobile platforms (presented in Section 2.1) can be detected with network traffic analysis methods.

Grayware Previous research by The Wall Street Journal [183] and Fulton [92] already demonstrates the great potential of in-depth network traffic analysis to identify behaviour typical for grayware (see Section 3.2.6.2). Especially grayware with aggressive advertising practices, for example, transmitting unique identifiers together with the location of the user to advertising companies, can be discovered. Since grayware usually does not devote that much effort to cover its habits, inspection methods with *direct matching* are already capable to reveal transmission of sensitive information over the network.

In-depth network traffic analysis helps expert users to identify applications showing characteristic typical for grayware and a rich representation of the result might help to raise the awareness of the user about privacy concerns outgoing from such applications.

Furthermore, reports and statistics about things like the contacted domains and servers, gathered during the general network traffic analysis, allow to recognise patterns in the network traffic common for grayware.

Malware The methods used to identify grayware can be applied in the same manner to discover malware. However, the different aim of malware and the thereby deployed methods, with more sophisticated techniques to obscure and hide the behaviour, limit the detection capabilities of network traffic analysis. Still, in-depth traffic analysis might uncover in some cases malicious activities, where private data is exposed over the network.

The insight into the traffic gained with general traffic analysis could help to discover communication between malware and C&C servers or reveal the misuse of a device for DDoS attacks.

A comprehensive tool, providing automated analysis of the network traffic, could help to detect the different threats arising from Android applications. Since modern devices contain a multitude of private information and can reveal a lot about the user, it is especially of interest to identify the exposure of sensitive data. The combination of general and in-depth network traffic analysis in a tool could help expert users to efficiently analyse a large number of applications and allow to quickly assess the risk outgoing from an application. The next chapter will present our developed tool NF4Droid, which aims to exactly fulfil this task.

Chapter 5

Network Traffic Analysis with NF4Droid

The previous chapter highlighted various aspects affirming the potential of specialised network traffic analysis for Android applications. As part of this thesis we developed a tool called Network Forensics for Android (NF4Droid) with the aim to provide exactly such adapted analysis functionality for the network traffic captured from Android applications. NF4Droid is a comprehensive tool, implemented as desktop web application which provides basic functionality like test archiving and management. It, furthermore, supplies in-depth analysis capabilities and offers features like the rich presentation and visualisation of the test results. The tool is meant for security experts and provides assistance in the analysis of application behaviour and the detection of malicious applications based on the information from the network traffic.

In this chapter we will present all the steps involved in the workflow of NF4Droid as illustrated in Figure 5.1. First, we will present methods for the capturing of the network traffic with external tools. Secondly, we describe the whole data processing involved with NF4Droid, starting at the data import, followed by the further processing and completed with the in-depth analysis. Finally, we present the selected approach for the intuitive and rich presentation and visualisation of the network traffic and the analysis results.

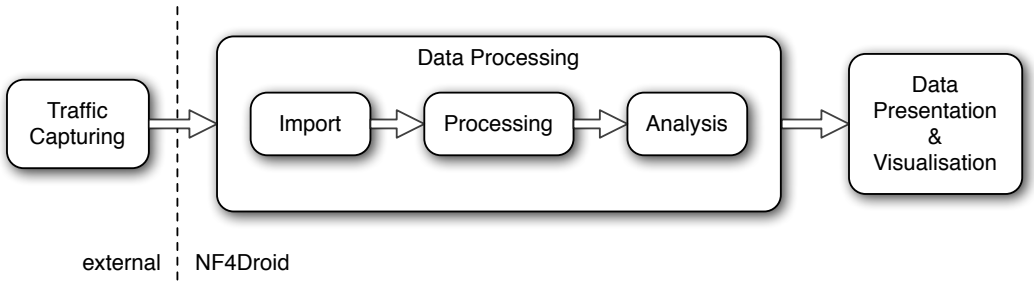

Figure 5.1: Graphic, illustrating the workflow of NF4Droid, consisting of three main steps: First, the capturing of the network traffic (external, not part of NF4Droid). Next, the data processing with the subtasks import, processing and analysis for the preparation of the data. Finally, the presentation and visualisation of the gathered data.

5.1 Traffic Capturing (external)

The Figure 5.2 highlights the traffic capturing step of the overall workflow of NF4Droid, discussed subsequently. NF4Droid centres upon the analysis of packet capture (PCAP)-files to be independent of

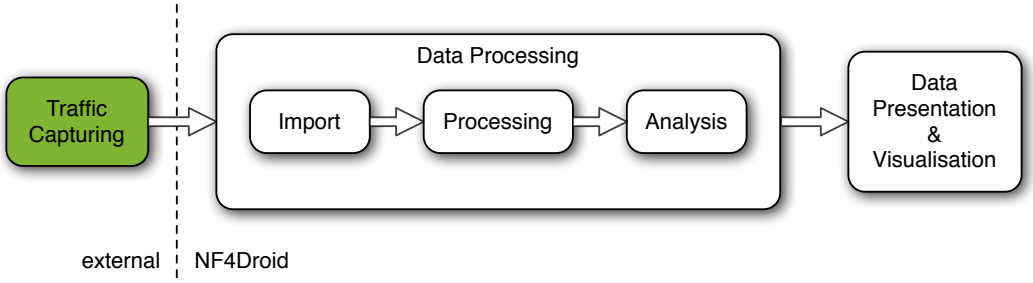

Figure 5.2: Graphic, presenting the workflow of NF4Droid highlighting the first, network traffic capturing, step. The traffic capturing is actually not done by NF4Droid as illustrated in the graphic, but still part of the overall process.

the underlying capturing method. Accordingly, the actual process of capturing the network traffic during the run of an application and the automation of the test process is not part of NF4Droid.

In Section 3.2.6.1 we already briefly described some of the existing network traffic capturing methods. Now we explain them more detailed. Additionally, we highlight the problems of automated testing and the associated simulation of user interaction and point out, why we need to record the test environment properties.

5.1.1 Capturing Methods

In the following we describe common network traffic capturing methods deployed for Android devices and highlight their advantages and restrictions.

5.1.1.1 tcpdump

tcpdump [179] is a multi-platform command-line packet analyser, which includes functionality to capture the network traffic and store it into PCAP-files. Cross-compiled versions of the tool are available for Android, which allow to directly capture the traffic on the device and enable to easily automate the capturing process. Through the execution on the device it is even possible to capture the traffic, no matter which network interface (e.g. 3G or Wi-Fi) is used. However, the execution requires root permissions and is thus only applicable on rooted devices.

5.1.1.2 Emulator

The Android emulator can be instructed with the command-line parameter

```
emulator -tcpdump /path/to/output-file.pcap
```

to directly capture the traffic and write it to a PCAP file. With this method you do not have to bother about the use of the tcpdump tool since the emulator has a dump tool integrated, which is automatically instructed.

5.1.1.3 VPN service

Since the version 4.0 the Android API offers the possibility to integrate custom VPN services [70] what can be utilised to directly capture the network traffic on standard devices without requiring root access.

tPacketCapture [54] is an Android application providing exactly such capturing functionality. Nevertheless, capture capabilities are limited with this method to the Internet Protocol and no traffic information about the link layer is available [54, 70], but similar to tcpdump, it does not matter which network interface is used.

5.1.1.4 Wi-Fi setup

With this method the device is connected to a special set up Wi-Fi where all the produced traffic is captured with common network traffic capturing tools like tcpdump [179] or Wireshark [191]. Therefore, an advantage of this method is that it does not pose any restrictions on the device and the deployed system. Not only no root permissions or certain system versions are required, the method is even independent of the deployed OS. Moreover, the external setup allows to apply advanced tools and methods for traffic interception and manipulation, especially interesting for man-in-the-middle attacks on secured communication (see Chapter 5.2.3.4). A drawback is the more complicated test setup which complicates automated testing. Furthermore, sophisticated malware might only show its malicious behaviour if it is connected over the 3G network. Accordingly, that could not be detected with this method.

Every capturing method has different requirements on the test system and all methods vary in their possibilities to be instructed during automated testing. According to deployed system and the used approach for test automation, the most suitable method should be selected.

5.1.2 Automation and User Interaction

For thorough testing of an application it is important to reach and trigger every functionality provided by the application. Especially malware might only show its malicious behaviour at a certain point or after some time of the program execution. A manual test of an application by an human is certainly the most thorough test methodology. However, manual testing is a time-consuming process and thus limits the number of applications that can be tested.

While automated application testing allows to analyse a large number of applications it requires best possible simulation of the user interaction for thorough testing. As mentioned in Chapter 3 different approaches for the simulation of user interaction exist ranging from random user input known as monkey testing to code instrumentation based on the information from the static analysis. Additionally to the automation of the application testing, it is important for us that a network traffic capturing method can be integrated in this process, which delivers traffic captures for the analysis with NF4Droid.

5.1.3 Test Environment Properties

For the in-depth analysis of the network traffic with NF4Droid (described in the subsequent Section 5.2.3) we require certain information about the environment of the tested system. Properties of the test environment like unique identifiers, the phone number or the device location are essential for our in-depth analysis and must be logged with the test. In general, the more information we have about the tested system, the more aspects can be analysed. Thus, for thorough analysis with NF4Droid it is important to not only capture the network traffic, but also log the environment properties. In Section 5.2.3.1 we enumerate such properties present on Android devices which should be logged with the traffic capture.

The capturing of the network traffic and the test automation is a wide-ranging topic which is left to other projects. NF4Droid focuses on the analysis of the traffic captures produced by such methods and only states which properties need to be logged for the in-depth analysis.

5.2 Data Processing

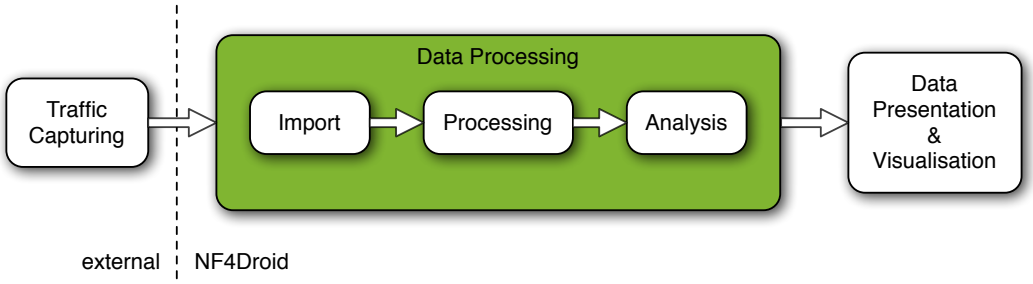

Figure 5.3: Graphic, presenting the current data processing step in the overall workflow of NF4Droid, consisting of the subtasks: import, processing and analysis.

For the later use in the data presentation and visualisation of NF4Droid, the traffic capture files and the logged environment properties must be imported into the platform and further processed. In the following we will describe the subtasks import, processing and analysis of data processing stage in the overall workflow illustrated in Figure 5.3.

5.2.1 Import

The first step after the capturing process is the upload of the traffic capture file onto the NF4Droid platform together with the information about the according test environment properties. The Screenshot 5.4 shows the view of NF4Droid for this step in the workflow.

Since NF4Droid offers *test management and archiving* functionalities, a traffic capture must be associated to a certain application and application version. The user must enter the *application package name* (unique for every application) and the related *application name*. As assistance for the user, while typing the application package name a autocomplete feature will present possible matches of applications already available in the database (see Screenshot 5.4) for selection. Further on, the information about the exact application version, consisting of the *application version code* (specific for a certain application version) and the related *application version name*, must be entered. As with the application package name, an autocomplete feature is offered for the application version code.

Additionally, the user should enter the test environment properties related to the traffic capture. This information is essential for the in-depth analysis of data exposure with NF4Droid. The preselection and enumeration of the environment properties, required for the analysis, facilitates the work of the expert user, and it outlines which environment properties are important and required for the analysis. However, without this data NF4Droid can not apply in-depth analysis and thus only provide general information about the network traffic.

5.2.2 Processing

For the in-depth analysis and for the rich presentation and visualisation of the traffic data, further processing of the information from the traffic capture and the test environment is required. The data must be structured and enhanced, to be suitable for the various analysis and easily retrievable for advanced representations.

As a first step the uploaded PCAP-file must be *parsed*, *enriched* with additional data and *persisted* in a object model together with the information about the application and the test environment properties.

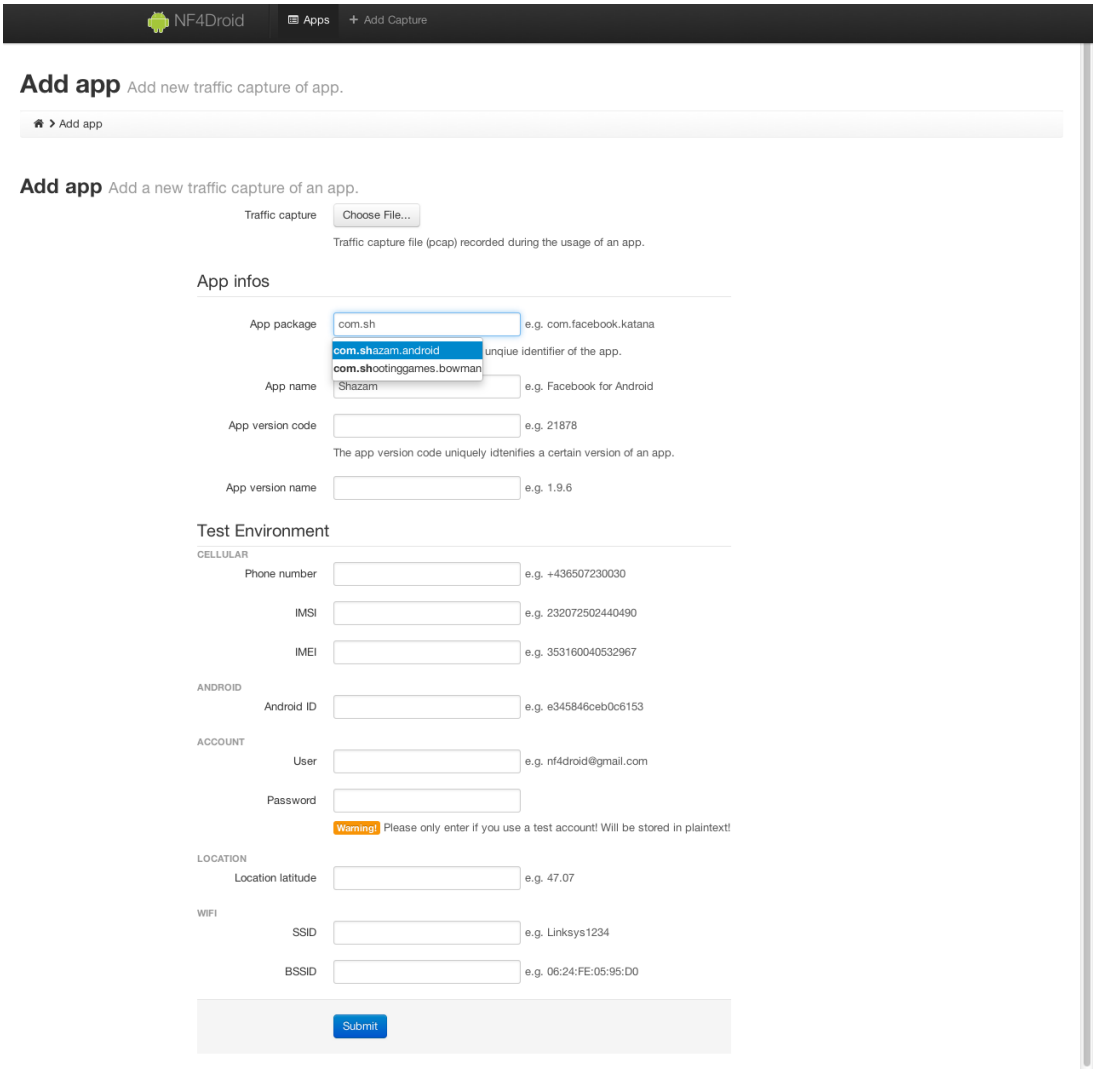

Figure 5.4: Screenshot of the view for the import of traffic captures and the according test environment properties in NF4Droid. As further detail, the autocomplete feature, which presents possible matches of existing application package names available in the database for selection, is visible in the screenshot.

Subsequently, can the *in-depth analysis* run various tests on this rehashed data. In the following we present this processing steps in greater detail.

5.2.2.1 Parsing

The traffic capture files must be provided in the PCAP format used by tcpdump any many other network traffic capturing tools. PCAP-files represent the complete captured network traffic in a format that allows to reconstruct every single part of the communication. By parsing the file the whole information about each data unit from all involved network layers, independent from the utilised communication technology and the involved network protocols, can be extracted for further processing.

This comprehensive information about the communication of an application builds the data source for the in-depth analysis and the presentation and visualisation part of NF4Droid. To model structural connections and relevant characteristics of the network traffic information, maps NF4Droid the information gained from the parsed traffic capture file into its own data model. This data model reflects the network traffic information in a form easy to access and use in the in-depth analysis process and which

allows to easily retrieve the information for various statistical representations.

5.2.2.2 Enrichment

To provide extensive information about the network traffic, NF4Droid enriches the basic data from the traffic capture with various additional data. For example, NF4Droid makes use of IP geolocation databases to provide information about the location of the servers contacted by an application. Besides, we extract data like the host part of contacted Uniform Resource Locators (URLs) to be able to create certain statistics. Moreover, we make the presented information easier to understand for the user by supplying additional facts. For instance, we denote the underlying service, belonging to a particular port number.

5.2.2.3 Persisting

NF4Droid follows the strategy to keep all the imported and aggregated data stored in a centralised database, where each traffic capture is associated with a certain application and version of an application. This approach enables quick data access during the in-depth analysis and allows to run comprehensive queries on a large set of information in the data presentation and visualisation step. Moreover, it is easy to compare analysis results from multiple traffic captures of an application or from different application versions.

Additionally, the storage of all traffic information might allow to run newly added analysis on previously tested applications. Similarly, future enhancements of the data presentation and visualisation parts can directly utilise the large set of existing information.

The parsing, enrichment and persisting steps are crucial in the preprocessing of the data for the later use in the in-depth analysis and data presentation and visualisation. Only a well-conceived model for the large amount of data as deployed in NF4Droid makes quick access and retrieval of the important information possible.

5.2.3 In-Depth Analysis

The in-depth analysis of the transmitted data plays a key role in the network traffic analysis of Android applications with NF4Droid. The network is the main interface of today's devices to the outerworld. Accordingly, all most all information leaving the device has to pass through this point. This fact emphasises the potential to identify exposure of sensitive information in the network traffic. Hence, the thorough analysis of the transmitted data helps to identify unwanted or even malicious behaviour of applications.

To know about which information should be considered during the analysis it is important to be aware about what information is actually present on modern devices. As part of this thesis we tried to identify the most important data generally present on devices running the Android platform. From this comprehensive list we selected those information, considered sensitive or private enough and, furthermore, analysable with our deployed methods for the in-depth analysis.

Applications might use different protocols for the communication over the network. However, to be able to identify certain information in the transmitted data, the in-depth analysis method needs to be adapted for every protocol. Currently, NF4Droid supports the commonly deployed HTTP, where we analyse the parameters and header fields of the HTTP requests for the transmission of sensitive data.

The analysis method deployed in NF4Droid relies on the test environment properties, provided together with the traffic capture, to identify exposure of information. The pursued method simply searches for occurrences of the environment properties in the parameters and header fields of the HTTP requests of the network traffic. However, applications commonly use data obfuscations methods like data hashing to protect or hide information. Accordingly, the differently represented information could not be

detected with the simple analysis method. To counteract this problem, the analysis of NF4Droid tries to imitate the obfuscation methods to achieve improved results, for example, by applying the same hashing functions on the original data.

In the following we first present the list of information, which we might consider during the in-depth analysis. Next, we describe the deployed analysis method in greater detail and provide a overview table for the currently implemented properties we support in the in-depth analysis. Finally, we point out certain limitations of the analysis method.

5.2.3.1 What to look out for?

To answer the question "What to look out for?" during the analysis of data exposure it has to be generally considered, what data is present on modern devices and how sensitive it is.

One starting-point is the security architecture of the Android platform. It is, beside others security measures, primarily is built upon a permission system which restricts the access to sensitive parts. It is designed in a manner that no application has, by default, permissions to perform any operations that would adversely impact other applications, the operating system, or the user [31]. Therefore every application, which wants to access the user's private data (such as contacts or text messages) or certain hardware (e.g. camera) has to state up front the required permissions. At install-time the user can decide whether to grant the application all the demanded permissions or to not install it at all [81].

By examining the list of available permissions in the Android SDK [28] it is already possible to enumerate a large number of critical data, which it is worth to look at during the analysis. For example, the `ACCESS_FINE_LOCATION` permission, which allows an application to enquire the position using the GPS-sensor, leads to the idea to check for transmitted GPS-coordinates [28]. Permissions like `READ_PHONE_STATE` need further investigation to identify the accessible data. By studying all the Java classes of the Android SDK (which are related to the permission), it can be determined that among other things the phone number can be obtained (see Paragraph II.).

Besides the data which is protected by permissions, it is even of interest to investigate the usage of other generally available data. The reference of the Android SDK helps to identify which data is present and how it can be accessed. This leads to the investigation of things like the information about the system or data located on the external storage.

Furthermore, it is important to always check for all possible exposures of data, even if an application does not have permission to access it, since research showed that there are ways to circumvent the permission system. Security researchers created a proof-of-concept zero-permission[1] application, which can gather data located one the external storage, a list of all installed applications and some basic information about the system [156, 157]. Additionally, the application uses a trick to transmit the whole information to a server without a direct permission to access the Internet (although certain limitations exist).

The following summary of data, available on a Android device, should give an overview and help to select important information, which should be taken into account during an in-depth analysis of data exposure. Although the list is quite extensive it might still not be complete and for simplification it is limited to the typical environment for Central Europe. Slight differences might exist for countries where other technologies are more established. For example, Global System for Mobile Communications (GSM) versus Code Division Multiple Access (CDMA) based mobile networks.

I. System Information about the hardware and software of the device.

Android ID [22] A 64-bit quantity, that is generated and stored when the device first boots or a factory reset is made, which is commonly used as unique device identifier. However, it is not completely

[1] An application which requests and requires no permissions.

reliable as unique identifier at older Android SDK versions and due to bugs from major device manufacturers [184].
e.g. e345846ceb0c6153

Locale [17] A language/country combination used to select the regional representation of information like numbers or dates.
e.g. en_GB/English/GB, de_DE/German/Germany

Manufacturer / Brand / Product / Model / Device [9] Information about the manufacturer of the product/hardware, the brand (e.g., carrier) the software is customised for, the name of the overall product, the end-user-visible name for the end product and the name of the industrial design.
e.g. Samsung / Google / Nexus S / Nexus S / crespo

Android SDK version [10] Build version of the installed *Android SDK* represented by a user-visible version string, a corresponding integer constant or a internal value used by the underlying source control system to identify a certain build.
e.g. 4.0.3, 15 (for 4.0.3), IML74K (for 4.0.3)

Kernel version [25] OS kernel version.
e.g. 3.0.8-g6656123

Radio firmware version (Baseband version) [9] The version string for the radio firmware (often called baseband).
e.g. P729BB01

Serial number [9] A hardware serial number for the device unique within the manufacturer.
e.g. 90016442894F00FC

Identifiers are of special interest for advertisers and even attackers since they allow to almost uniquely identify a device or even user. If an identifier can be aggregated with personal information about the device owner and data like its location it is possible to track the user and analysis his behaviour an habits.

Furthermore, might information about the hardware and the installed software version give attackers hints about possible security vulnerabilities they can exploit [112].

II. Cellular Information about the cellular module of the device and the connection established with it [26].

Phone Number The *phone number* of the Subscriber Identity Module (SIM) card also referenced as *Mobile Subscriber Integrated Services Digital Network Number (MSISDN)* consisting of a *C&C* (e.g. 43), *National Destination Code (NDC)* (e.g. 650) and *Subscriber Number (SN)* (e.g. 7230030) [122].
e.g. +436507230030

Required permissions: `READ_PHONE_STATE`

Mobile Country Code (MCC) Unique identifier specifying the mobile network country [123].
e.g. 232 (for Austria)

Mobile Network Code (MNC) Unique identifier used to specify a carrier within a country [123].
e.g. 03 (for T-Mobile in Austria)

Location Area Code (LAC) Unique identifier which specifies a group of GSM cells within an *Location Area (LA)* [123]. *MCC*, *MNC* and *LAC* form together the *Location Area Identity (LAI)*.
e.g. 2520

Required permissions: `ACCESS_COARSE_LOCATION` or `ACCESS_FINE_LOCATION`

Cell-ID (CID) Unique number identifying a GSM cell within an *LA* [185]. *LAI* and *CID* combined allow to globally identify GSM cells [185].
e.g. 46929

Required permissions: `ACCESS_COARSE_LOCATION` or `ACCESS_FINE_LOCATION`

Network Country Network country as *International Organization for Standardization (ISO) 3166-2* [121] code derived from the *MCC*.
e.g. AT (for Austria)

Service Provider Name (SPN) Network carrier name.
e.g. T-Mobile Austria

Required permissions: `READ_PHONE_STATE`

International Mobile Equipment Identity (IMEI) A 15-digit number associated to a device to uniquely identify it internationally as a mobile station. It is allocated by the manufacturer and registered by the network operator which maintains a *Equipment Identity Register (EIR)*. Network operators have the capability to maintain "blacklists" of *IMEIs* to block the access to the network, for example, if the device got stolen [185].
e.g. 353160040532967

Required permissions: `READ_PHONE_STATE`

International Mobile Subscriber Identity (IMSI) The *IMSI* consists of the *MCC*, *MNC* and *Mobile Subscriber Identification Number (MSIN)* and is stored on the SIM card. Each user/contract can be internationally uniquely identified by this number. To access a network the *IMSI* (together with the *IMEI*) will be verified by the network operator [185].
e.g. 232032502440490

Required permissions: `READ_PHONE_STATE`

SIM serial number Unique serial number of the SIM card often referenced as *Integrated Circuit Card Identifier (ICCID)*.
e.g. 89430700001180095600

Required permissions: `READ_PHONE_STATE`

Voice-mail number Number which is called to retrieve voice-mails (sometimes includes the phone number).
e.g. +43650117230030

Required permissions: `READ_PHONE_STATE`

The IMEI is often used by developers, advertisers and attackers as unique identifier for a device. Furthermore, the phone number and IMSI allow to uniquely identify a user. Therefore, such unique identifiers can be used to track users and thus pose a risk for the user's privacy.

In addition, the voice-mail number (and if not included the phone number) can together with the IMSI be used by attackers to access voice-mail recordings using a method called "Caller ID Spoofing" [45, 66].

III. Wi-Fi Information about the Wi-Fi module of the device and the connection established with it [27].

Required permissions: `ACCESS_WIFI_STATE`

Service Set Identification (SSID) The name of a Wireless Local Area Network (WLAN).
e.g. nf4droid

Basic Service set identification (BSSID) Usually the Media Access Control (MAC)-Address of the Access Point (AP) used to uniquely identify the WLAN.
e.g. 06:24:FE:05:95:D0

IP Address Internal IP address assigned to the device.
e.g. 192.168.179.15/24

MAC Address MAC-address of the device Wi-Fi network interface.
e.g. 00:24:fe:24:e7:62

The BSSID could be used to obtain coarse location of the device by querying a online geo location service (e.g. Google Location Server (GLS) [101], Skyhook Location [170]), independently of the integrated location API of Android (see section V.), and thus approximately locate a device without any permission regarding location services.

IV. Network Information about the currently connected network regardless of the connection type (Wi-Fi, 3G) [19].

Required permissions: `INTERNET`

IP Address IP address assigned to the device.
e.g. 192.168.179.15/24

MAC Address MAC-address of the currently used network interface of the device.
e.g. 00:24:fe:24:e7:62

V. Location Depending on the availability of hardware, the granted permissions and the desired accuracy, speed and battery-efficiency the following methods can be used to obtain the location of a device [18, 20]:

Cell-ID (CID) Google maintains a crowd-sourced database of mobile network CIDs with according GPS coordinates [124]. By querying the database for the CID of the currently connected network station an approximate position can be retrieved from the GLS [101].

Required permissions: `ACCESS_COARSE_LOCATION` (or `ACCESS_FINE_LOCATION`) and `INTERNET`

Wi-Fi A similar database, like for the Cell-ID (CID), exists for the BSSID of Access Points [1]. Querying the database for the information of the connected/nearby Wi-Fi can provide approximate location information.

Required permissions: `ACCESS_COARSE_LOCATION` (or `ACCESS_FINE_LOCATION`) and `INTERNET`

GPS / A-GPS The Global Positioning System (GPS) uses satellites to provide precise location data. Assisted GPS (A-GPS) speeds-up the positioning progress.

Required permissions: `ACCESS_FINE_LOCATION` and `INTERNET` (for A-GPS)

The enumerated positioning methods may yield to the following data [43]:

Latitude Geographic coordinate that specifies the north-south position of a point on the earth's surface in degree. Values: Degrees from -90 (south) to 90 (north).

Longitude Geographic coordinate that specifies the east-west position of a point on the earth's surface in degree. Values: Degrees from -180 to 180, positive values represent the eastern hemisphere.

Altitude True altitude describing the elevation above the mean sea level in meters.

Bearing Direction of travel in degrees east of true north. Values: Degrees from -180 to 180, 90 degrees for east, -90 degrees for west and +180 or -180 for south.

Speed Speed of the device over ground in meters per second.

Accuracy Accuracy of the location information in meters.

Location is a very valuable information for advertisers since it allows location-aware advertisement. The location information gets even more valuable if you can correlate it to a certain device or user and maybe track it over time [73, 108]. Since modern smartphone devices are our daily companions and often carried around with us all the time, they can give a deep insight in our life and expose various sensitive information. For example, it can reveal where you live and work, where you usually buy your groceries, how often you go to the fitness centre and a good deal more. This shows that it is of great interest to investigate if the location is used and which information is send over the network together with the location, especially information, which could identify us as a certain user like unique identifiers.

VI. Accounts Android offers a centralised registry for the management of the user's online accounts [4]. The user enters credentials (username and password) once per account and simple grants application permission to access them using a "one-click" approval. By a modular concept a variety of authentication methods can be supported. Commonly servers support authentication tokens to authenticate requests to the server without sending the user's actual password. The account management can create such tokens directly for applications, thus applications do not need to handle passwords themselves.

Name Login name for the account.
e.g. nf4droid@gmail.com

Required permissions: `GET_ACCOUNTS`

Type Account type defining the underlying service.
e.g. com.google, com.facebook.auth.login, com.skype.contacts.sync

Authentication Token Authentication token (auth token) for the specified account type and a particular account. The user has to enter the credentials if no entry exists yet. If a previously generated, still valid, auth token exists it is directly returned. Otherwise a new auth token will be requested from the corresponding server. If a saved password is available it will be used for the request, otherwise the user is prompted to enter his password.

Required permissions: `USE_CREDENTIALS`

User Metadata Arbitrary additional data stored with the account used for things like the full name of the user.
e.g. Joe Bloggs

Required permissions: `AUTHENTICATE_ACCOUNTS`
Additionally to the permission the password will only be available to an application which has the same Unique User Identifier (UID) as the one which authenticated (created) this account [16].

Password The password of the account.
e.g. 123456

Required permissions: `AUTHENTICATE_ACCOUNTS`
Additionally to the permission the password will only be available to an application which has the same UID as the one which authenticated (created) this account [16].

VII. Calendar Calendars available on the device ranging from subscribed/synchronised private and public calendars to locally stored ones [11].

Required permissions: `READ_CALENDAR`

Title and Description Title and description for the event.
e.g. “Meeting with Joe Bloggs”, “Release of SuperPhone 7”

Event Organiser Email Address Email address of the event organiser.
e.g. nf4droid@gmail.com

Start/End Date/Time, Duration, Repeated, Time zone Information about the date and time when the event occurs, how long it takes and which time zone is referenced. For repeating events (e.g birthdays) the interval.
e.g. 15/05/2012 9:12am (GMT) every year

Location Location where the event takes place.
e.g. “Office of Mr. Bloggs”, “1600 Amphitheatre Parkway, Mountain View, CA 94043”

Reminders Additional reminders for an event.
e.g. Alert 10 minutes before

Attendees Persons invited to the event and their according information (Name, Email,...) and attendance status.
e.g. Joe Bloggs (joe@bloggs.com, speaker, required) - declined

Calendars are used to organise our daily lives and give a deep insight in our schedule. Additionally, they might include business information which should be kept private (e.g. the release of a new product).

VIII. Contacts / Profile Information obtainable for all contacts on the device and the user profile of the owner [13]. The information might by synchronised with multiple servers (Google, Skype,...) and aggregated from various external sources (Facebook, Google+, Skype...).

Required permissions: `READ_CONTACTS` (for contacts), `READ_CONTACTS` and `READ_PROFILE` (for the user profile)

Name (Given name, Family name, Title,...) Data representing the contacts proper name.
e.g. Bloggs John Joe, Ph.D.

Nickname Nickname of the contact.
e.g. Joy

Email addresses Email addresses (work, private,...) of the contact.
e.g. joe@bloggs.com

Groups Groups to which the contact is assigned.
e.g. Co-worker, family

Instant messenger addresses (ICQ, Skype,...) Number or user name for an instant messaging service.
e.g. joe123, 12312311

Notes Personal notes for the contact.
e.g. Met first on DEFCON 2011.

Organisation (Company, Department, Office location,...) Information about the organisation the contact works for.
e.g. Pear Inc., CEO, Room E231

Phone numbers (Mobile, Home, Work, FAX,...) Phone numbers of the contact.
e.g. +436507230030

Photo Profile picture of the contact.

SIP addresses Voice over IP telephony address based on Session Initiation Protocol (SIP).
e.g. +18012345678@sip.voice.google.com

Postal addresses Addresses (work, home,...) of the contact containing information like country, region, city, postcode, street (with house number), post office box number.
e.g. 1600 Amphitheatre Parkway, Mountain View, CA 94043

Website Website of the contact.
e.g. www.bloggs.com

Relation Relation to the contact.
e.g. Father, Brother, Domestic Partner

Events (Birthday, Anniversary,...) Events related to the contact.
e.g. Born 15/05/1985

Social Stream data Data (status updates, likes,...) aggregated from social streams of the contact.
e.g. Joe likes android.com

Required permissions: `READ_SOCIAL_STREAM`

Our profile and the list of our contacts can reveal a lot of information ranging from phone numbers to addresses. Exposure of the contacts means not only exposing our own private data but also information about our family, friends or co-workers. It is important to consider carefully which applications should be allowed to access this information and even more it might be of interest to evaluate where this data is send to if an application has the permission for it.

IX. Bookmarks and Browser History Bookmarks and the history of visited websites of the integrated web browser of Android [8].

Required permissions: `READ_HISTORY_BOOKMARKS`

URL The Uniform Resource Locator (URL) of the website.

Title The title of the website or for a bookmark the user entered title.

Number of visits Counter for the number of visits for this URL.

Last visited Date and time when this URL was last visited.

Created Date and time when this entry was created.

The bookmarks and the browsing history can reveal a lot information about our interests and habits and, moreover, the might include personal stuff like certain search terms we used.

X. Android Logs [30] Logs are used by developers to help identifying bugs during debugging. The following categories of logs exists for Android [74]:

Main The main log for applications.

Events For system events information.

Radio For radio and phone-related information.

System For low-level system messages and debugging.

Required permissions: `READ_LOGS`

Research showed that sensitive data like the browsing history, SMS, contacts and location were accessible through logs which included to much detail [132]. Therefore, the permission to read logs allowed to access information which should actually be protected by other permissions.

XI. Call Log [12] The history of phone numbers which have been called.

Required permissions: `READ_CONTACTS`

Number The phone number which was called.

Date/Time The date and time when the number was called.

Duration The duration of the call.

Type Defines if the call was incoming, outgoing or missed.

Similar to the contacts this information gives insight about with whom we stay in contact. For example, it might be undesired for a business man that somebody knows about the companies he stays in contact with.

XII. SMS / MMS Short Message (SMS) or Multimedia Message (MMS) stored on the device [24].

Required permissions: `READ_SMS` or `RECEIVE_SMS` (for intercepting incoming SMS) or `RECEIVE_MMS` (for intercepting incoming MMS)

Body (Message, Data) Body of the message containing the text and additionally files (pictures, sounds,. . .) for MMS.

Subject Message subject (only for MMS).

Originating/Recipient address Phone number of the originator or recipient.

Type (Sent, Received, Draft,...) Defines if the message was sent, was received, is a draft,. . . .

Date/Time Date and time when the message was sent/received/created.

SMS and MMS often include very personal information like the messages you send to your partner or pictures you send to some friends. Thus this information might be considered highly private.

XIII. Files / Data Android offers primarily the following possibilities to store data on the device [15]:

Internal Storage Stores arbitrary data on the internal memory of the device. By default (and with `MODE_PRIVATE`) all files stored to the internal storage are private to the application and no other application can access them (nor can the user). By using the `MODE_WORLD_READABLE` or `MODE_WORLD_WRITEABLE` during the file creation it will be accessible by everybody. When an application gets uninstalled all files stored on the internal storage are removed.

External Storage The shared external storage, which can be a removable storage media like an SD-card or an internal (non-removable) storage, can be used to store any desired data. However, the data is not protected and can be accessed by all applications and the user.

Shared Preferences Used to store primitive data like booleans, floats, ints, longs, and strings as key-value-pairs. Similar to the internal storage the data can just be made accessible to the application itself by using `MODE_PRIVATE` and shared with others by using `MODE_WORLD_READABLE`, `MODE_WORLD_WRITABLE` or `MODE_MULTI_PROCESS`.

SQLDatabase A SQLite[175] database private for an application to store structured data and easily query it.

Shared preferences and the SQLDatabase are mainly used to store smaller amounts of data like settings and textual-data whereas internal- and external storage can be used to store larger data and especially files. Since the internal storage is often limited in space, files like pictures, music or downloads are commonly stored on the external storage.

Since no permission management is available for the external storage, the data is easily accessible by every application and thus developer should consider carefully what to store at this place. To emphasise what this means, any arbitrary application can access images, stored on the external storage, *without any permission* [48]. Additionally, if the application has the permission to access the Internet, it can even upload it in the background to some server.

XIV. Applications A list of all installed applications/packages can easily be retrieved (without permission) and yields to the following information [7, 21]:

Package Name The Java package name which serves as a unique identifier for the application.

Version Code / Name The version number and name of the package.

Unique User Identifier (UID) The assigned kernel UID which is used to run the application and to enforce filesystem permissions. Might not be unique if it is a shared UID which allows multiple applications to share certain data or run in the same process. However, in this case the applications have to be signed with the same signature.

Install Time / Last Updated Date and time when the application was installed and last updated.

Permissions The requested permissions of the application.

Providers Declared content providers that supply structured access to data managed by the application [14].

Activities Declared activities which implement parts of the application's visual user interface.

Features Hardware or software features used by the application.

Services Declared services which are used to implement long-running background operations or communication APIs.

Signature The signature of the application signed with the developer account and used to identify the author of the application and to, furthermore, establish trust relationships between applications [23].

Data Directory Full path to a directory assigned to the package for its persistent data.

Source Directory The full path to the location of this package.

This information might help attackers to get knowledge about which data is available on a device if certain applications are installed and which permissions are granted to an application. For example, it is more likely to find pictures on a phone where an application with the permission to access the camera is

installed. Moreover, information about the software version of installed applications could lead attackers to security vulnerabilities they can utilise. Additionally, attackers might even be interested to scan the accessible directories of an application for files which include sensitive data.

XV. Email Since the mail application is not directly integrated into the Android SDK, there is no API, which allows direct access to mails. Mail clients store mails by themselves and it underlies to the application to provide a content-provider to access them.

However, mails often include a lot of private or business information and it is thus important to protect the included information. Since the protection depends on the deployed mail application it is of interest to inspect if any information included in the mails leave the device without the knowledge of the user.

XVI. Conclusion This enumeration present a comprehensive list of data available on Android devices and helps to identify the environment properties for the in-depth analysis. However, for the in-depth analysis not every information is equally interesting and important. Furthermore, the information differs in the complexity and ability to be detected with the deployed in-depth analysis methods. For example, more sophisticated methods are required to identify the leakage of pictures than the transmission of unique identifiers.

5.2.3.2 Analysis Method

The large amount of information, available on modern devices and the quite varying format of the data, puts different demands on the analysis method for the identification of the information in the network traffic. The currently implemented in-depth analysis of NF4Droid pursues the straightforward approach to apply a textual search for the occurrence of certain information in the header fields and parameters of the HTTP requests. This simple, but still quite effective method is not applicable for all types of information, but particular suited for number-sequences or textual-information.

The analysis method uses the values from the test environment properties, logged together with the traffic capture and searches for occurrences in the HTTP requests. However, information like unique identifiers or passwords are commonly obfuscated, using data modifications and hash methods before they are transmitted over the network. Such data manipulations would make it already impossible to detect the information with the textual search. Still, to overcome this drawback NF4Droid tries to imitate the obfuscation methods by applying the same modifications on the properties, given from the test environment. In the moment NF4Droid supports the following frequently used hashing methods: MD5, SHA1 and SHA26. Additionally, we try to imitate more specific obfuscation methods. For example, the AdWhirl [2] advertising company appends a constant string to the Android ID before hashing it, leading to a different hash value. To still be able to detect the exposure of the Android ID, we imitate the exact same behaviour. Similar, we always generate the hash value of the lower- and upper-case version of a string property, since they lead to different hash values.

If the exposure of a property from the test environment is detected during the network traffic analysis, it is persisted in database together with the information about the according HTTP request it has been exposed in and the applied obfuscation method for the subsequent presentation and visualisation with NF4Droid.

Although the analysis method of NF4Droid has certain limitations (see Chapter 5.2.3.4), it already leads to considerable results. NF4Droid especially shows its strength in the detection of unique identifiers. Still, the analysis is designed to be applied on different test environment properties and the currently integrated test are listed in the following section.

5.2.3.3 Currently Analysed Test Environment Properties

The large number of information present on today's devices makes an analysis for all properties very difficult. NF4Droid focus mainly on the analysis of those properties, which are concerning the privacy of the user and are easily detectable with the deployed analysis method. Currently, only a small number of tests are included in the analysis which, already lead to good results relating to heavy advertising habits of applications (see Chapter 7). Still, the goal is to add further tests, thus NF4Droid is designed in such a way that it can quite easily be extended with further analysis tests.

Property Type	**Exposure Point**		**Data Obscuring**			
	Parameter	Header Fields	Plaintext	MD5	SHA1	SHA256
Android ID	x	x	x	x	x	x
IMEI	x	x	x	x	x	x
IMSI	x	x	x	x	x	x
Location	x	x	x	-	-	-
Phone number	x	x	x	-	-	-
User name	x	x	x	-	-	-
Password	x	x	x	x	x	x
SSID	x	x	x	-	-	-
BSSID	x	x	x	-	-	-

Table 5.1: Overview of the currently analysed test environment properties in NF4Droid, including information about the point, where the analysis is applied in the HTTP request and which obfuscation methods are imitated.

The Table 5.1 shows the different property types from the test environment, which are currently tested during the in-depth analysis with NF4Droid. These properties have been selected since they are commonly used for advertising purposes, reveal the position of the device and concern the privacy of the user. Furthermore, they are quite easily detectable with the deployed analysis method. For every property we search for the exposure in the HTTP requests parameters and header fields. Moreover, we not only search for the exact same value (plaintext), but also for different hash values of the original value if it might be interesting for a property.

5.2.3.4 Analysis Limitations

Although the straightforward analysis approach deployed in NF4Droid leads to considerable good results (see Chapter 7), it has certain limitations. Currently, we only search for occurrences of certain properties in the parameters and header fields of HTTP requests. For a more comprehensive analysis the method should not only analyse the parameters and header fields of HTTP requests, but also the transmitted body content like files. Moreover, the in-depth analysis should cover more different network protocols, which could be used by malicious applications.

Besides the limitation on the places we search for exposure of information, even the way we try to identify the information in the transmitted data has certain limitations. NF4Droid directly searches for occurrences of the environment properties in the parameters and header fields. However, if the information is formatted differently, slightly changed or even obfuscated it might not be detectable with the deployed analysis method. We try to lower this impact by imitating obfuscation methods, especially the hashing of values. Still, we are not able to cover all possible obfuscation techniques and thus certain data exposure might remain undetected. Moreover, certain information is better suited for the analysis than other, for example, the direct search is not so qualified for long texts as for short and specific strings, since slight changes in a long text would make it already undetectable.

Furthermore, encryption of data or the use of secured communication protocols like HTTPS makes the detection with the deployed analysis method impossible. However, for HTTPS it might, depending on the quality of the protocol implementation in the application, be possible to apply a man-in-the-middle attack if the *Wi-Fi setup* capturing method is used. With this technique the intercepting point makes independent connections with the application and the servers and relays messages between them, thus making them believe the communicate directly with each other. At the intercepting point the unencrypted network traffic can thereby be captured and accordingly analysed with NF4Droid. Nevertheless, proper and secure implemented applications will not trust the intercepting point and refuse the connection.

Another problem we are facing is the mistakenly identification of information exposure. For example, if we search for a certain property, the same value might, without any relation to the actual value, occur in the transmitted data. In that case it would be incorrectly marked as data exposure. Therefore, we always denote that we found only a *possible* data exposure and leave the final decision about the correctness to the user.

Similarly, detected data exposure might correlate to actual wanted and required transmission of information of an application. For example, if you use a location based service the information about the position of the device has to be transmitted. Again, we leave the decision about, if the transmission of certain information is justified, to the user. Nevertheless, NF4Droid offers the possibility to identify to which severs the information is actually transmitted and thus assists the user in the decision.

The currently implemented in-depth analysis method of NF4Droid has limitations and there is certainly room for improvement. Still, the method is with its quite simple approach achieving quite good results and already able to reveal interesting facts about the exposure of information as presented in the Chapter 7. Additionally, NF4Droid is designed to be in the future extended with further tests and different analysis methods.

5.3 Data Presentation & Visualisation

To understand the behaviour of an application and to fast and easily identify malicious application, NF4Droid provides a rich presentation and visualisation of the information gathered and processed in the previous steps. With tables for the detailed examination of the information, filter and search capabilities to inspect certain aspects and various graphical visualisations of the comprehensive amount of information, NF4Droid aims to provide a solid network traffic analysis platform, specialised for the traffic of Android applications and the exposure of information.

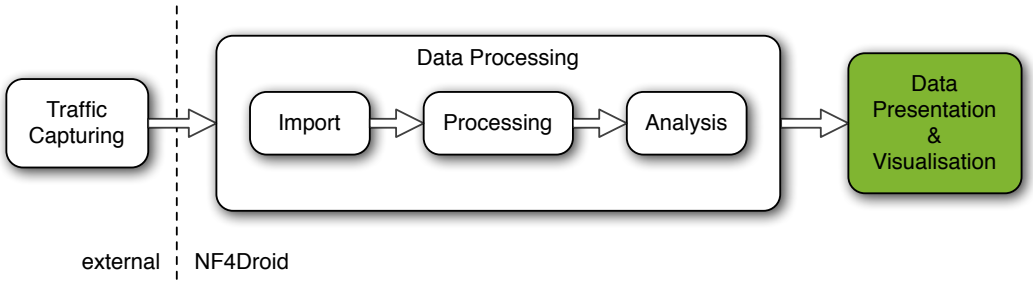

Figure 5.5: Graphic, illustrating the workflow of NF4Droid with the final, data visualisation and presentation, step highlighted.

The Figure 5.5 highlights the final data presentation and visualisation step in the workflow of NF4Droid. In the following, we present all the user interfaces, their according functionalities and their use in the overall analysis process, for this step. First, the general management and archiving user interfaces. Sec-

ondly, the overview dashboard presenting the first test results and acting as starting point for the more detailed analysis. Next, the traffic time line as comprehensive visualisation of the network traffic and the data exposure. Thereafter, the visualisation for the location of the contacted servers. Finally, the tabular and statistic information about the HTTP requests.

5.3.1 Capture Management and Archiving

NF4Droid offers functionality to manage and archive traffic captures of applications. Every imported traffic capture is assigned to the according application and application version for later retrieval. This allows to compare different traffic captures from different applications or different versions and captures of the same application. Furthermore, future versions of NF4Droid can run new analysis on the existing data or even taken multiple applications into account during the analysis.

Subsequently, we present the management sites of NF4Droid, which allows to hierarchically browse through the traffic captures of the applications.

5.3.1.1 Apps

The “Apps” site shown in Figure 5.6 presents the list of all applications, for which traffic captures exist in the database and allows to browse through the applications. Furthermore, the integrated incremental search enables to search for application and package names to present only matching entries in the table (see search for the term “google” in Figure 5.6). By clicking on an entry you are forwarded to next site, where the different application versions are presented for the selected application.

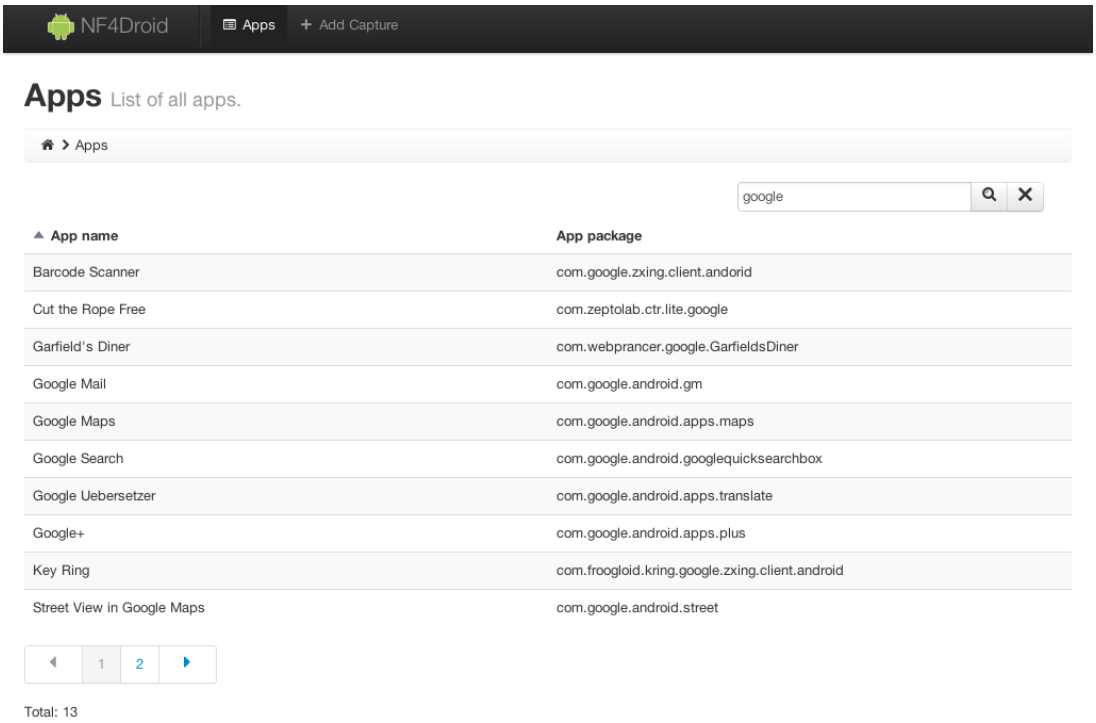

Figure 5.6: Screenshot of the “Apps” site in NF4Droid presenting the list of applications for the search term “google”.

5.3.1.2 App versions

The “App Versions” site as presented in Figure 5.7 lists all versions available in the database for the application selected one the “Apps” site. The incremental search can be used to filter for certain application version codes or names. By selecting an application version you are directed to the “Traffic Captures” site containing all traffic captures for the specific version.

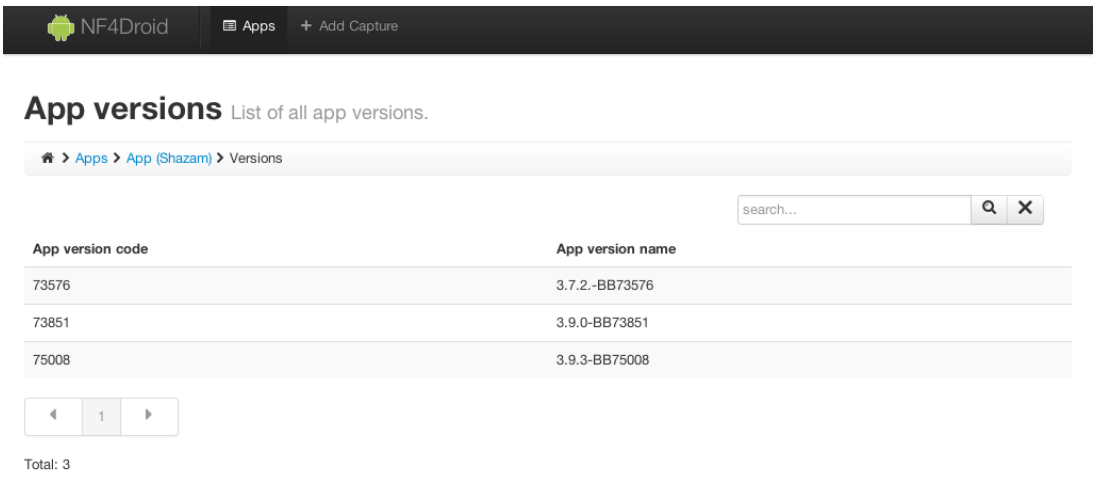

Figure 5.7: Screenshot of the “App Versions” site in NF4Droid presenting the list of versions for the “shazam” application.

5.3.1.3 Traffic captures

The “Traffic Captures” site shown in Figure 5.8 presents the traffic captures, stored in the database for a specific application and version of an application. The incremental search can be used to filter for traffic captures from a certain date or where the description matches the search term. By clicking on a traffic capture you are forwarded to the according “Capture Dashborad” site.

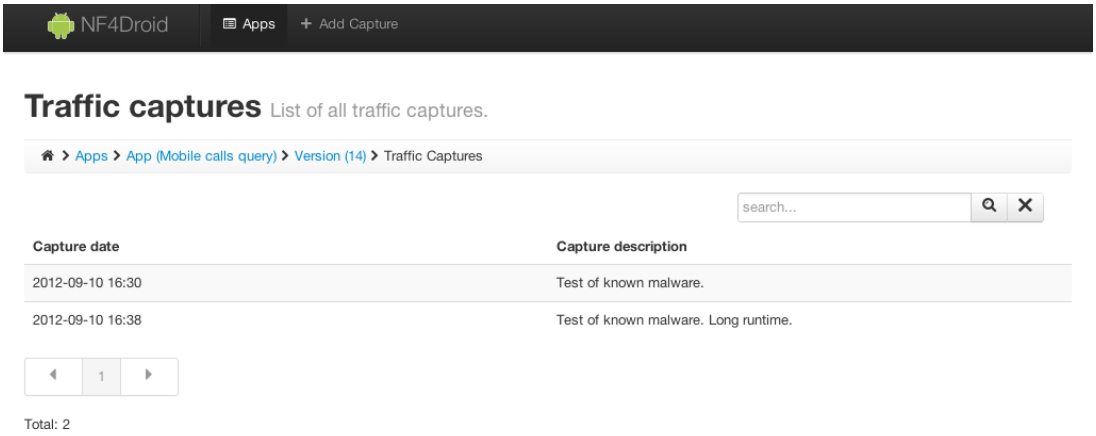

Figure 5.8: Screenshot of the “Traffic Captures” site NF4Droid listing all traffic captures for the “Mobile calls query” application with the version code “14”.

The implemented user interfaces for the management of archived traffic captures allows to fast an easily browse through the available traffic captures and facilitate the organisation of a large test base.

5.3.2 Capture Dashboard

The "Capture Dashboard" site shown in Figure 5.9 presents the first traffic capture specific information and acts as starting point for the further analysis by the user. It consists of the three following sections:

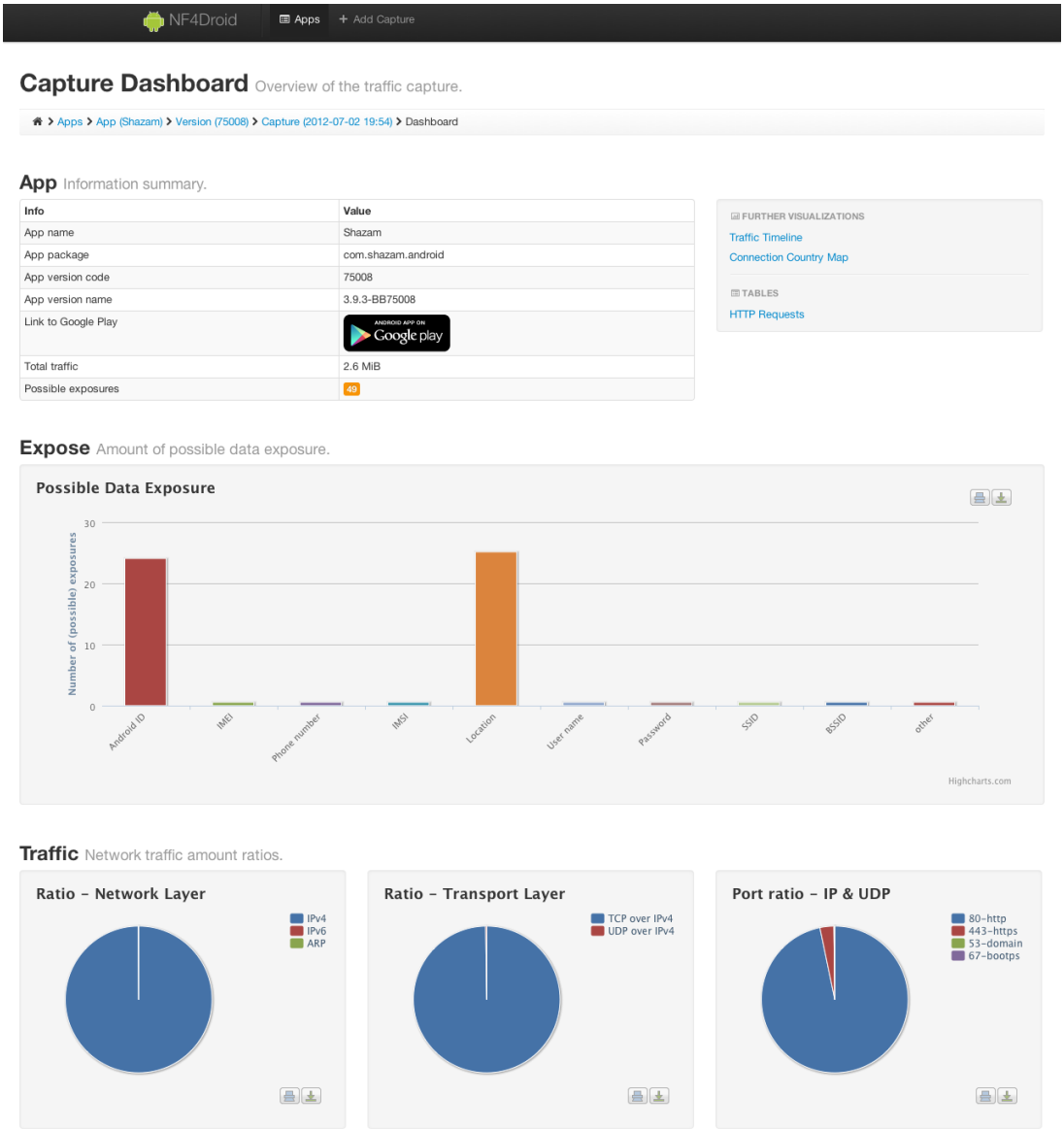

Figure 5.9: Screenshot of the "Capture Dashboard" site in NF4Droid consisting of three sections, namely "App", "Exposure" and "Traffic" for the information presentation and visualisation. Results are presented for the traffic capture of the "shazam" application.

"App" Gives a general summary about the application tested in this capture. It outlines the information about the application and application version, offers a link to Google Play and presents the *total amount of traffic* produced by the application and the *number of possible data exposures*.

The information about the total traffic amount, could help the user in the application analysis process, to already reveal malicious application causing suspicious or inappropriate high traffic. Moreover, the total number of possible exposures indicates the potential risk outgoing from an application.

Additionally, this section includes the links to further visualisations and tables for the more detailed inspection.

“Exposure” Consists of a bar chart illustrating the number of data exposures per test environment property. This visualisations should make at first sight visible, if and how much information is exposed by an application. The data for the visualisations comes from the in-depth analysis during the data processing step of NF4Droid.

“Traffic” Presents general statistical information about the traffic type ratio per layer or protocol as pie chart based on the traffic amount. The left chart denotes the ratio on the network layer (e.g. IPv4, IPv6), the chart in the centre presents the ratio on the transport layer (e.g. Transmission Control Protocol (TCP) over IPv4, User Datagram Protocol (UDP) over IPv6) and the right chart shows the ratio of the used TCP and UDP ports. This information should help the user to find out which protocols are generally deployed by the application, and it especially reveals ,if an application uses secure communication protocols like HTTPS.

The “Dashboard” is the starting point for the analysis and should help to quickly get a first impression of an application and build the knowledge base for the detailed analysis on the sites containing more detailed information and further visualisations.

5.3.3 Traffic Timeline

The “Traffic Timeline” site shown in Figure 5.10 is one of the main analysis views in NF4Droid. It consists of following two sections, one for the graphical visualisations of the network traffic and data exposure, and the other for the tabular representation of the data exposure information:

“Traffic Timeline” Presents the produced traffic amount over the time of the application execution in an area chart. Initially, the series for the total traffic amount and for the amount of the HTTP requests are displayed. This information allows to identify exactly at which point of the application execution how much traffic is produced. This knowledge could, for example, be used to figure out, which part of an application must be accessed, that certain malicious behaviour is shown.

Additionally, it is possible to add series underlying certain criteria by clicking on the “Add filtered series” button. A modal box, as shown in Figure 5.11, will pop up for the definition of the criteria for the new series. Afterwards, the new series will be added to the existing chart to allow comparison of different series. The possibility to compare different series should assist the user in the identification of certain aspects, for example, how much traffic is coming from a certain IP and going to a specific country. In addition, every series can be deactivated or activated for better visibility in the chart, and manually added series can even be removed.

Another very important functionality is the visualisation of the data exposure in the “Traffic Timeline”. For every exposed information a flag marker is added to HTTP requests series at the exact point of the application execution, when the data was exposed. For better visibility every exposure type gets its own series in a different colour and the flags include a letter denoting the exposure type. By selecting one of the markers, detailed information about the data exposure and the related HTTP request is presented in a modal box, as shown in Figure 5.12. This intuitive illustration of the data exposure enables to directly recognise what an application is doing during its execution and should help to identify malicious behaviour.

“Exposure” Additionally to the graphical presentation of the information exposure in the area chart we provide a tabular overview of the data exposures. This direct data representation allows to provide more detailed information about the exposure like the actual value and the deployed obscuring method on first sight.

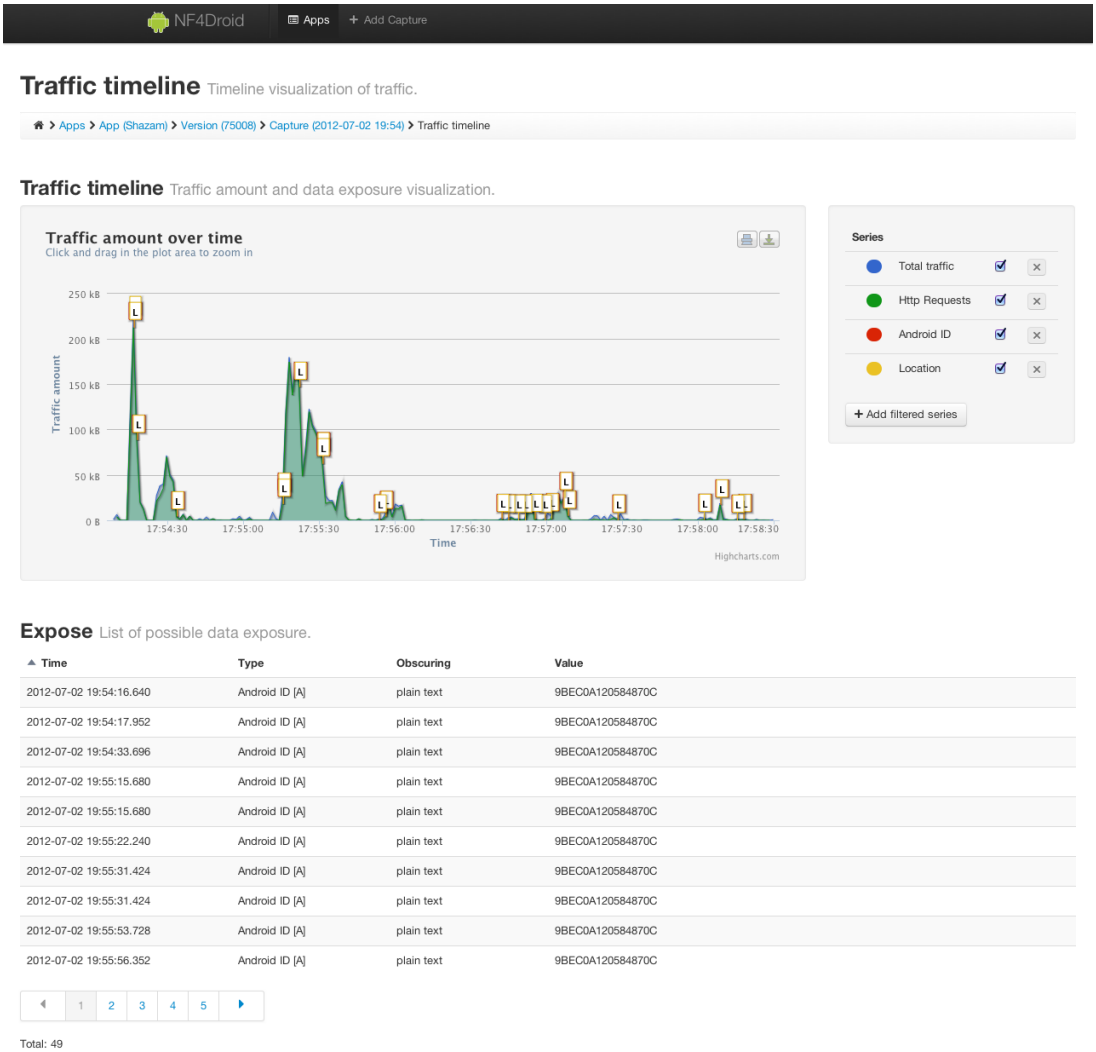

Figure 5.10: Screenshot of the “Traffic Timeline” site in NF4Droid consisting of the two sections “Traffic Timeline” and “Exposure”. The data shown is from an traffic capture of the “shazam” application.

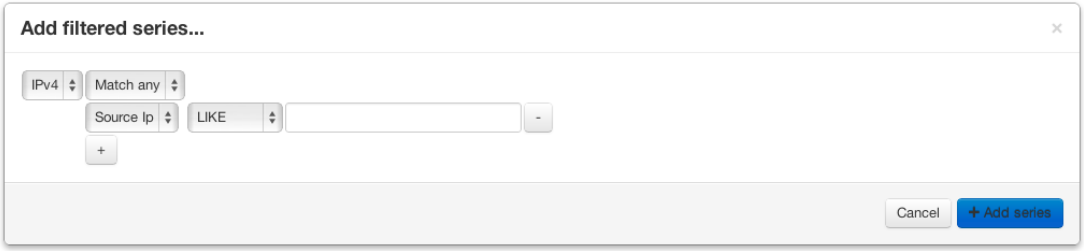

Figure 5.11: Screenshot of the modal box for adding a new series underlying certain criteria to the “Traffic Timeline”.

The “Traffic Timeline” provides a rich presentation of the traffic information in relation to the application execution. Moreover, it intuitively illustrates data exposure in a graphical and tabular representation. Hence, this view, with the interactive filter possibilities, should assist the user in the analysis of the application behaviour and detection of malware.

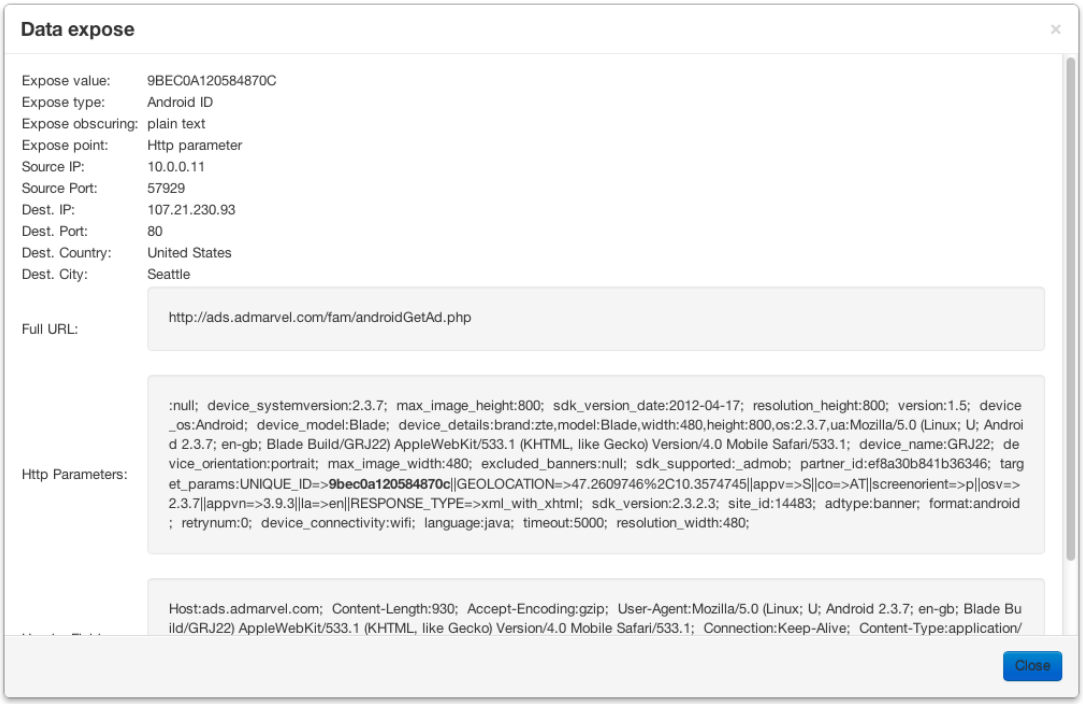

Figure 5.12: Screenshot of the modal box presenting detailed information about a data exposure and the related HTTP request. This specific entry points out the exposure of the Android ID in the HTTP request parameter by the “shazam” application.

5.3.4 Traffic Geochart

The “Traffic Geochart” site shown in Figure 5.13 presents the geographical distribution of the IPv4 network traffic in a geo chart. As mentioned in section 5.2.2.2 does NF4Droid lookup the location of every IP in an geolocation database. Based on this information, we present the distribution of the traffic amount, coming from and going to every country, in the chart. Additionally, we present the total IPv4 traffic amount and the amount of traffic for which we were not able to identify the location. This statistical information should give the user a general overview of the location of the servers contacted by an application.

5.3.5 HTTP Requests

The “HTTP Requests” site shown in Figure 5.14 is another comprehensive view for the detailed analysis of the HTTP requests traffic with NF4Droid. It consists of two interacting sections where the upper part of the view is a tabular representation of the HTTP requests and the section below presents statistical information about the corresponding HTTP requests.

“HTTP Requests” table The tabular presentation of the HTTP requests directly displays important facts like the URL, remote IP or the country. Moreover, detailed information about an HTTP request will be presented in a modal box, as shown in Figure 5.15, after selecting an entry. Besides that the view allows to browse and search for specific HTTP requests, it offers the possibility to filter for HTTP requests exposing certain information. This filter capability is especially useful to thoroughly analyse all HTTP requests, which expose information to find out facts like, where the information was transmitted to or what else was transferred in the same request.

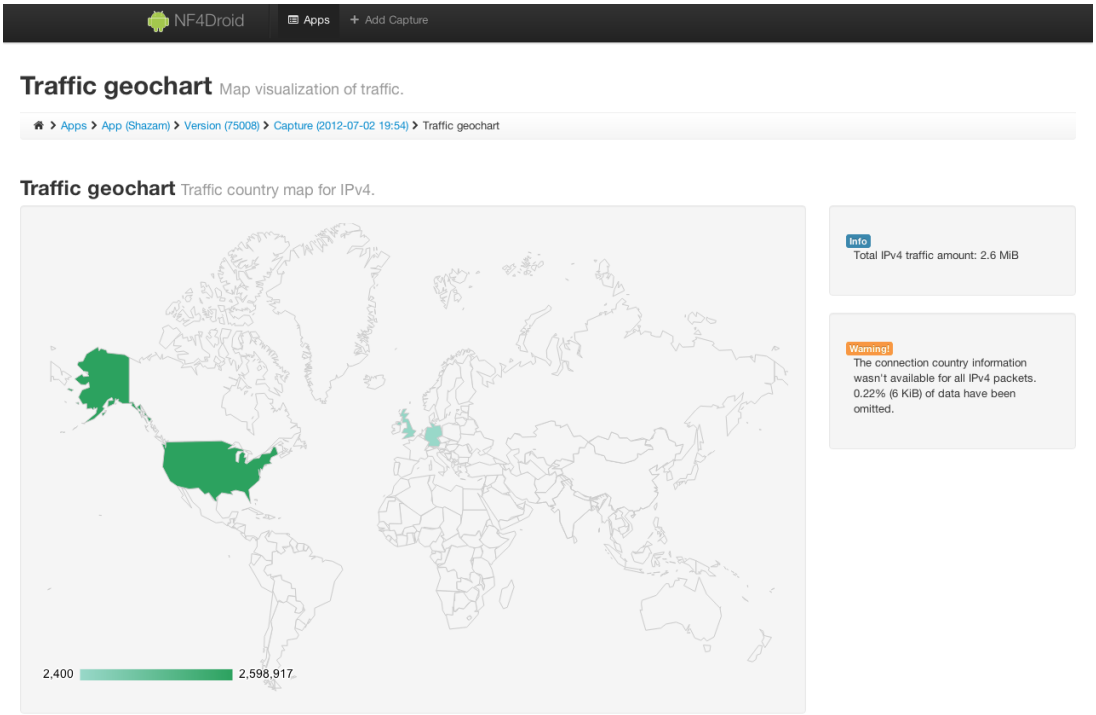

Figure 5.13: Screenshot of the "Traffic Geochart" site in NF4Droid presenting the geographical network traffic distribution for the IPv4 traffic. Results are presented for the traffic capture of the "shazam" application.

"HTTP Requests Statistics" In this section, statistic information about HTTP requests are presented in a bar chart. The ten most contacted URLs, hosts, remote IPs or countries can be visualised. Additionally, the filter from the table above even applies for the statistical visualisation enabling to provide specific results for certain exposure types. This statistic information could, for example, reveal URLs, which are very frequently contacted and thus suspicious. In the same manner, it might allow to identify servers of advertising companies, which are commonly contacted and transmit the location information.

NF4Droid provides a solid platform for the analysis of the network traffic from Android applications. It is independent of the deployed traffic capturing technique and it only requires the logging of certain test environment properties for the in-depth analysis. With a database in the background for the archiving of the further processed and enriched traffic capture information, it allows to run in-depth analysis for data exposure. Moreover, it is possible to run sophisticated queries on the data, for the rich presentation and visualisation of the traffic and exposure information. NF4Droid follows an elaborated strategy for the data presentation to allow experts to easily analyse the behaviour of applications and immediately recognise malicious behaviour. Therefore, we present in the Chapter 7 a case study to demonstrate the capabilities of NF4Droid. Prior, we will explain some implementation details of NF4Droid in the Chapter 6.

5.4 The Use of NF4Droid for Other Mobile Platforms

NF4Droid is currently build for the analysis of Android applications. One motivation was the fact that Android is currently one of the main targets for malicious applications (see Section 2.3). Additionally, at the Android platform advertising financed applications are commonly used, which concern the privacy

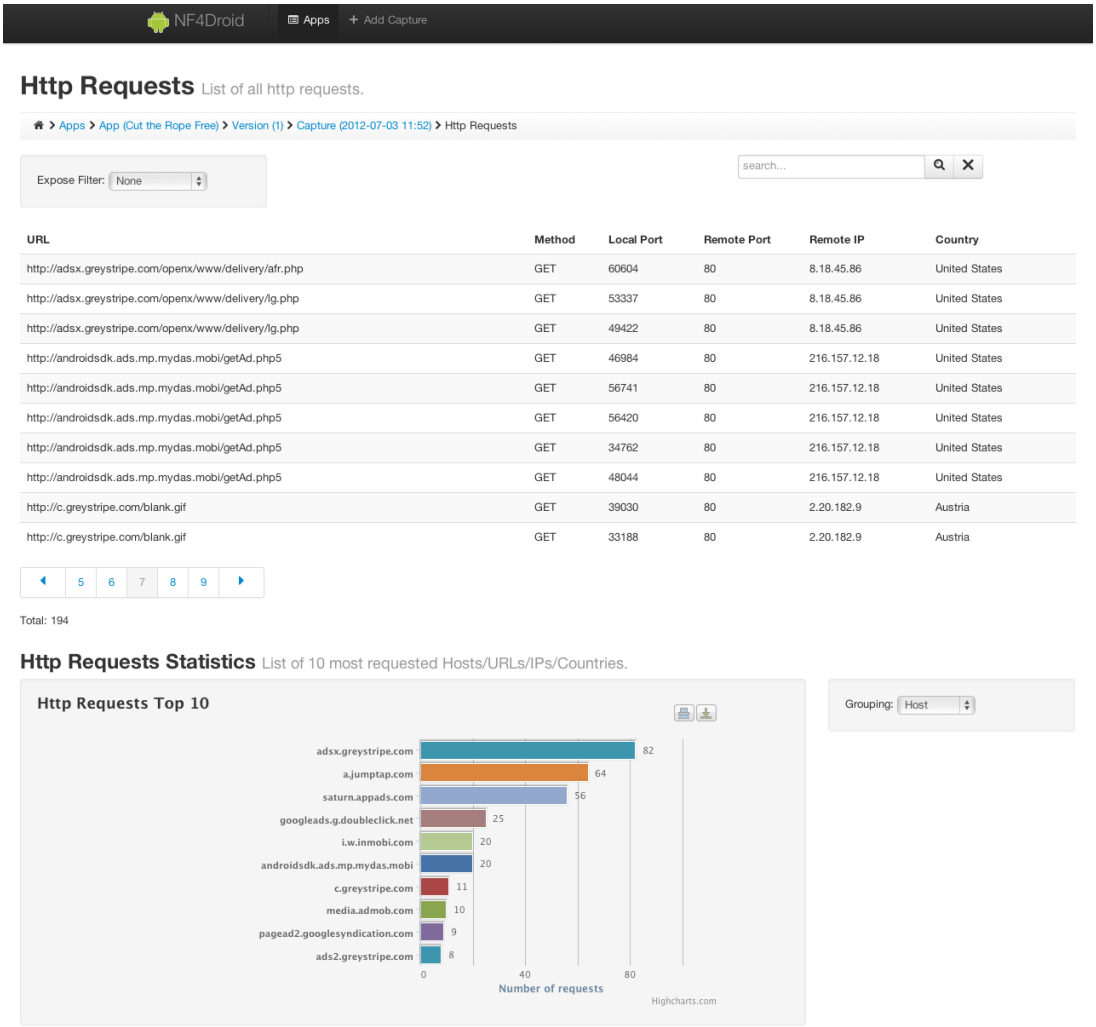

Figure 5.14: Screenshot of the “HTTP Requests” site in NF4Droid consisting of a tabular view for the HTTP requests with various filter and search possibilities and a section with statistical information about the HTTP requests. The data shown is from a traffic capture of the “Cut the Rope - free” application.

of the user. Accordingly, it is of great interest to analyse the behaviour of such applications. Another reason was that the Android platform offers more possibilities for the capturing of the network traffic, which are even easier to automate in the overall testing process (see Section 5.1).

However, the only platform specific parts of NF4Droid are the test environment properties, which we use for the in-depth analysis. Accordingly, to support other platforms, NF4Droid would only need to adapt the test environment properties to the one specific for the new platform. For example, the iOS platform uses different unique identifiers as Android. Furthermore, the deployed obfuscation methods might vary for different platforms and would need to be adapted.

Summarising, although NF4Droid currently focuses on the analysis of Android applications, the design would allow to easily adopt new platforms.

More details about the implementation and design of NF4Droid are given in the subsequent section.

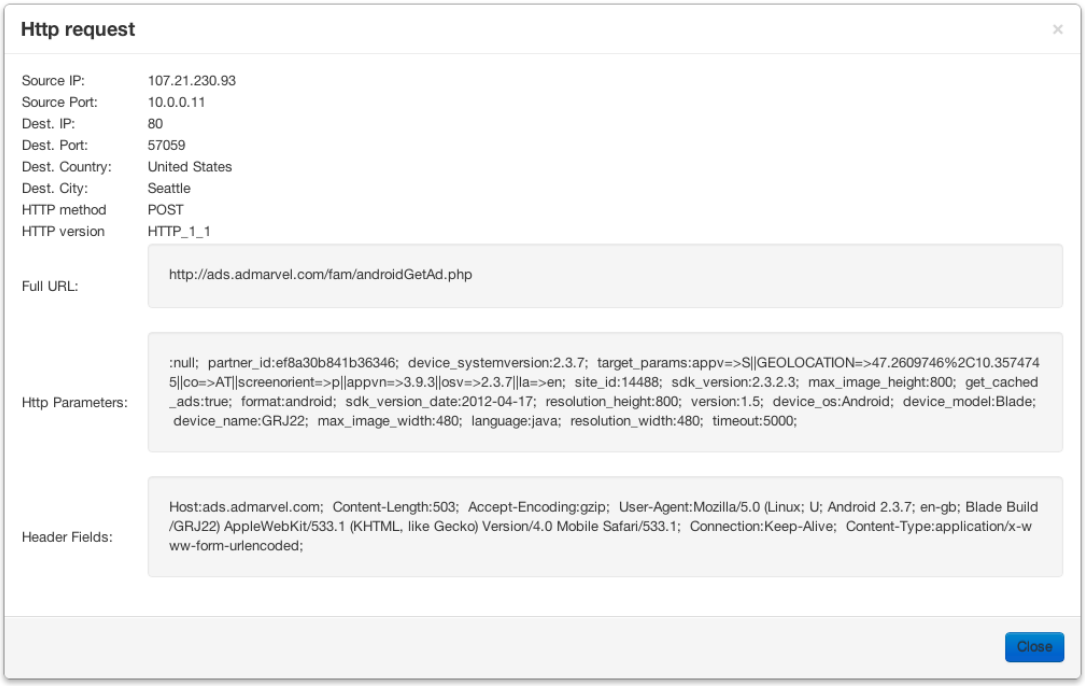

Figure 5.15: Screenshot of the modal box presenting detailed information about HTTP requests. This specific HTTP requests is part of the traffic captured from the “shazam” application.

Chapter 6

Implementation Details

This chapter discusses the implementation details of NF4Droid. We will outline the technologies and tools we used for the implementation and realisation of NF4Droid. Furthermore, we briefly describe the design and implementation of NF4Droid.

6.1 General Conceptual Design

NF4Droid is implemented as a *browser-based desktop web application*. The conceptual design to this approach is illustrated in Figure 6.1. Accordingly, NF4Droid consists of a *client side web application* for the data management, presentation and visualisation. Moreover, it provides a *server* component for the data processing and provisioning, which additionally persists all the gathered information in a *database* (DB) for archiving and fast data retrieval. The network traffic and test environment information, gathered during the external capturing, are imported into NF4Droid by the user. Subsequently, the network traffic information can be thoroughly analysed by the expert using NF4Droid.

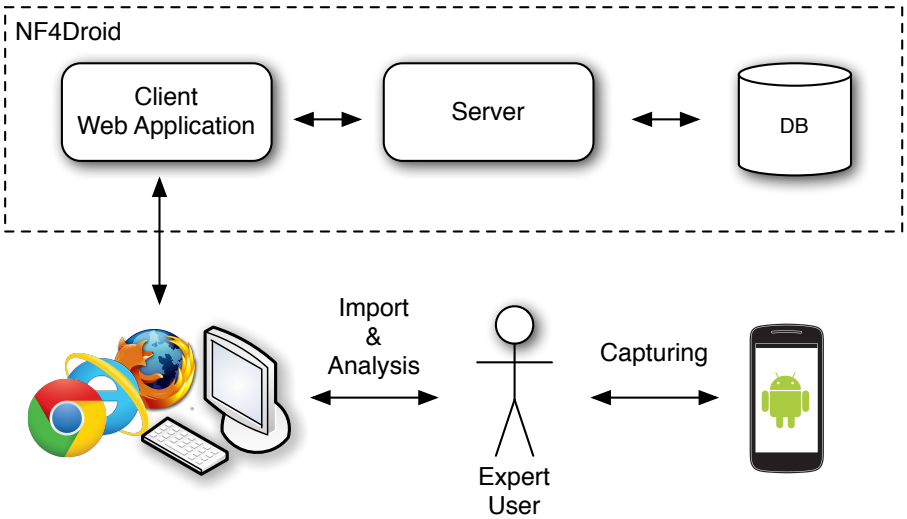

Figure 6.1: Graphic, illustrating the general conceptual design of NF4Droid. NF4Droid is designed as a desktop web application, consisting of a web based client, a server and a database. The user externally captures the network traffic of the Android applications and imports the gathered data to NF4Droid for the further analysis.

We choose the web application approach since it allows to provide the user a platform-independent

centralised analysis tool. One of the benefits is, that the user does not need to install anything on his computer to analyse the network traffic captured from Android applications. Furthermore, the centralised concept allows to easily update existing or integrate new features. Consequently, we can re-run the analysis on the existing data set. Another advantage is that, the browser-based web application can be used independent of the underlying OS and it runs on all modern browsers. However, it is optimised for the use with Google Chrome [100] version 20.0 or higher.

Although, NF4Droid is build as web application, it is not designed for the use on mobile devices. Mainly, because it has a different aim, which is the use as analysis tool by security experts. Accordingly, no optimised mobile version was developed, mainly because restrictions like small screen sizes would limit the possibilities for the data presentation and visualisation.

The *server side* stores all the imported traffic captures and the related test environment properties in a object-model persisted in a database. For this purpose, the imported PCAP-files are parsed and the thereby gathered information is further processed and subsequently transferred into our object-model. Subsequently, we apply in-depth analysis for exposure of data on the extensive set of information obtained during the import. Moreover, the server provides interfaces for the fast and easy retrieval of the acquired information for the rich presentation at the front-end.

The *client side* provides an intuitive and interactive UI for the representation of the information processed from the server. The rich presentation of the large amount of information for the thorough analysis by the expert user is achieved using interactive charts for the data visualisation and tables for the detailed enumeration of the extensive information. Additionally, the web application allows to add traffic captures together with the test environment properties to the server. Furthermore, it provides management functionality for the fast and easy retrieval of the archived traffic captures from the server.

In the following we present the design and the technologies, we pursued in NF4Droid for the tasks denoted above, in more detail.

6.2 Implementation of NF4Droid

In this section we describe how we implemented NF4Droid and which technologies we used for this task.

6.2.1 GWT based Web Application

For the implementation of the NF4Droid web application we used the GWT development toolkit (see Section 6.4.2). GWT allows to build advanced browser-based Asynchronous JavaScript and XML (AJAX) applications. GWT facilitates the development of web based application, since it allows the developer to code in purely object-oriented Java and the client side code is automatically translated into optimised JavaScript. Furthermore, it allows to easily communicate between the server and client side, providing different communication technologies. Additionally, it provides widgets and components for the creation of the web based UI.

6.2.1.1 Client-Server Communication

In NF4Droid we use two different approaches provided by GWT for the communication between the client and server side. Both allow to asynchronously communicate with the server, however, they vary in the level of abstraction and the underlying communication method.

On the on hand we use *GWT-Remote Procedure Call (RPC)* [106] for *service-oriented* communication. The basic building block of GWT-RPC are remote methods (similar to Java Remote Method Invocation (RMI) [152]). GWT-RPC provides automatically generated proxy classes at the client side for the server side service methods. It handles the transmission and serialisation of the the Java objects

passing back and forth as method parameters and return values. However, the transmitted data types have to either be *primitive data types* or classes that are *serialisable* (implement the "Serializable" interface) and provide a *default constructor*. Hence, it is possible to define arbitrary Data Transfer Objects (DTOs) which encapsulate certain information for the transmission between client and server.

On the other hand we use the *RequestFactory* [104] for the creation of *data-oriented* services. In contrast to GWT-RPC, RequestFactory takes a more prescriptive approach and uses domain entities and services as basic building block. It uses entity proxy interfaces for the client-side definition of server-side entities. RequestFactory implements its own protocol for data exchange between client and server. Additionally, it keeps track of objects that have been modified and sends only changes to the server. However, there are certain limitations on the data types that can be transmitted. The transmitted types must either be *primitive data types* or *entity proxy interfaces* itself.

For the direct use of our domain model classes at the client side we use the ReqeustFactory approach. However, for the visualisation of certain information we have services which provide none domain specific data, accordingly we transmit the data using GWT-RPC with custom DTOs for the payload.

6.2.1.2 Activities, Places and Model View Presenter (MVP)

In NF4Droid we use the *Activities and Places* framework [103] from GWT for browser history management, in conjunction with a slightly modified *MVP* design for the construction of the UI.

Activities represent actions performed by the user. Activities contain no UI specific code, but however typically restore a certain state, perform initialisation and load a corresponding UI. Activities are started and stopped by *ActivityManagers*, which are itself associated with a certain *Widget* of the UI. The ActivityManagers select the corresponding Activity based on the the mapping defined for *Places* in the *ActivityMapper*. Hence, a ActivityMapper is associated to a specific ActivityManager.

Places are objects representing a particular state of the UI. For example, they often include identifiers of the currently presented objects. Moreover, Places can be converted to and from a URL specific for the Place. This means, if a certain Place gets active (a URL gets called), the ActivitiyManager selects and starts the corresponding Activity, based on the mapping defined in the associated AcitivityMapper. Subsequently, the Activity initialises and loads its associated UI and presents it in the Widget associated with the ActivitiyManager.

With NF4Droid we tried to follow the MVP user interface design pattern with the aim to create a maintainable and established software structure and to follow the "separation of concerns" concept. In our case, the *model* consists of the domain model data, represented by the entity proxy interfaces of the RequestFactory and the DTOs from GWT-RPC. The *views* contains all of the UI components representing our application. However, the views do not directly interact with the model. The *presenter* provides the views with the information from the model. Furthermore, it handles the user interaction from the view and performs the required actions on the model. In contrast to the classical MVP concept, with a separate presenter, in NF4Droid the presenter is combined with the class implementing the Activity. Furthermore, our views do not handle certain events by themselves. The views provide interfaces for the registration of predefined handlers. Accordingly, the class implementing the presenter functionality can register itself to this events and directly implement the event handling.

6.2.1.3 User Interface (UI)

For the creation of the web technology based UI we used GWT's *UiBinder* [102] framework. UiBinder can be used to build UIs in a declarative manner using a mixture of XML and HyperText Markup Language (HTML). Certain XML tags allow to directly use the *Widgets* and *Components* provided with GWT. Moreover, attributes can be used within the layout to designate fields for the later binding with the related data types in the Java class. Accordingly, style definitions files are bind to corresponding

Java classes. This concept allows to separate the construction of the UI and the implementation of the application behaviour.

To create a intuitive UI and for a nice look and feel we used GWT-Bootstrap (see Section 6.4.11) in addition to the UI components of GWT. GWT-Bootstrap offers various great looking UI components and widgets. Moreover, it provides grid based layout mechanisms for the creation of structured web pages. GWT-Bootstrap can seamlessly be used together with GWT and it supports the UiBinder framework.

6.2.2 Rapid Application Development with Spring Roo

To fast and easily create the domain model for NF4Droid and for the generation of the initial GWT project setup, we used Spring Roo (see Section 6.4.1) as rapid application development tool. Spring Roo provides a command line tool where you can easily define your domain entities and the corresponding properties. Subsequently, the tool generates the according domain classes and the project setup. The project setup contains general boilerplate code and files required for the use of the domain classes in a GWT project and provides Maven (see Section 6.4.16) build files for the project compilation and deployment.

Additionally, AspectJ files (see Section 6.4.15) are generated which add functionality to the domain classes required for the ORM and the use within a persistence framework, based on the JPA (see Section 6.4.4) standard. Moreover, Spring Roo can automatically initialise Hibernate (see Section 6.4.3) as persistence framework for the ORM of the domain classes into a relational database. Furthermore, it creates all the therefore required entity management classes. Moreover, AspectJ files which extend the entities with basic methods, like for the retrieval from the persistence framework, are generated.

Besides, Spring Roo automatically generates the classes and interfaces used in GWT to communicate between the server side domain objects and the client side code, based on the GWT RequestFactory approach.

Although Spring Roo allowed to quickly create the initial GWT project following established design patterns, certain manual modifications of the generated code were required. It was especially necessary to optimise the mapping between the different classes in the domain model. Furthermore, we needed to extend the basic functionality, to allow more sophisticated queries on the data (see Section 6.2.4).

6.2.3 Network Traffic Information Import

To import and use the network traffic information gathered by the user we have to upload the PCAP-files to the server and subsequently extract the information from the files.

Since GWT does not provide file upload handling, we use the GWTUpload Servlet (see Section 6.4.6) for the upload of the PCAP-files to the server. GWTUpload is a Java Servlet providing functionality like file upload progress information and file handling on the server. It facilitates the integration of files upload in GWT based applications.

For the parsing of the PCAP-files we use the purely Java based Kraken library (see Section 6.4.7). Kraken allows to extract the complete network traffic information from the PCAP-files, including all packets (respectively segments, frames and data) for every layer of the Open Systems Interconnection (OSI)-Model [120]. Additionally, the library provides functionality to reconstruct the connection streams and sessions from the network traffic using Decoders.

The comprehensive network traffic information acquired in this processing steps builds the main data for NF4Droid. The data is subsequently used for all data analysis and visualisations.

6.2.4 Data Persistence, ORM and Data Retrieval

To have access to the extensive network traffic information captured for every application, we store the complete data acquired during the import in our domain model. The model is designed in a way that the traffic information is easily accessible for our in-depth analysis and quickly retrievable for the comprehensive data visualisations. Furthermore, we deploy an ORM on the domain model to persist the network traffic information in a relational database.

As persistence layer and for the ORM of the data we use Hibernate (see Section 6.4.3). It allows to easily map our domain model following the JPA standard and provides a solid persistence framework.

To more easily build the complex queries necessary for the data visualisation we use QueryDSL (see Section 6.4.5). QueryDSL provides a fluent API which allows to build type-safe syntactically valid queries, which are subsequently translated into JPA standard conform queries. All domain types and properties can be referenced safely and no direct Structured Query Language (SQL) is involved. Moreover, it is easy to incrementally define queries. Additionally, QueryDSL can directly uses the EntityManager provided by Hibernate to access the persisted data.

However, Hibernate has certain restrictions on the queries that can be executed on the mapped data. For example it does not support SQL like "Union All" queries. Since we require such specialised queries for some of our visualisations in NF4Droid, we have to use a workaround. QueryDSL provides a code generation feature which extracts the schema in the database and generates according Java classes [159]. This Java classes can subsequently be used to directly build native but type-safe SQL queries. Through the extraction of the model information from the database schema we achieve type-safety for the SQL queries, since the schema is generated by Hibernate according to the domain model and the ORM. However, a update of the domain model might require the regeneration of the related SQL query classes.

6.2.5 Further Processing

During the further processing we extend the traffic data by additional information like the geographical location of the IP address. For this purpose we use the MaxMind IP Database (see Section 6.4.9), which allows to approximately obtain the location of an IP address with an simple API and geo IP database files. The database files are freely available and updated every month. Hence, it is important to update the database files every month to achieve more accurate results.

Moreover, to provide the user with service names for TCP and UDP ports we use jnr-netdb (see Section 6.4.8) for the retrieval of the information. The further processing allows to provide additional informations and visualisation and makes the information easier to understand for the user.

6.2.6 Exposure Analysis

To analyse the network traffic for exposure of data, we directly search in the HTTP request parameters and header fields, extracted from the network traffic captures during the import. Like all traffic information, the HTTP request data is represented in the domain model and persisted in the database, so that it can quickly be retrieved. Accordingly, we directly search on the database over the domain model, using SQL based "LIKE" queries. In the HTTP request data we search for certain information using the test environment properties provided by the user together with the traffic capture.

To imitate certain obfuscation methods, data hashing methods are required. We directly use the Java implementations of the hashing algorithms, since they are as well used on the Android platform and accordingly deliver the same results.

To easily add new analysis we designed the tests in a abstract way, which makes definition of new test-cases very easy. To create a new test, the developer only has to implement an abstract class, which already provides basic functionality necessary for the analysis. In the new test class the user just has

to load the required test environment data from the model and define the analysis parameters. The parameters define which traffic information should be analysed (HTTP parameters or header fields) and which obfuscation methods (none, MD5, SHA1 or SHA-256) should be applied on the data during the test.

6.2.7 Data Visualisation

For the rich visualisation of the large amount of information we use different kind of charts and maps. To provide great looking and interactive charts we use GWT Highcharts (see Section 6.4.13) and Google Chart Tools (see Section 6.4.14). Both libraries provide different kinds of chart types and can be directly be used in GWT. We utilise pie charts to illustrate ratios between information like the used network protocols. Moreover, we use line charts to visualise the traffic amount over time. Additionally, we add flag markers denoting the exposure of information. Column bar charts are used to illustrate how much information of a certain type was exposed. Furthermore, geo charts represent the geographical distribution of the contacted servers.

This different visualisation support the security expert in the analysis of the network traffic information and make certain aspects easer to recognise.

6.2.8 Project Build

For the build automation and the management of external libraries we deployed Maven (see Section 6.4.16). In the XML based Maven Project Object Model (POM) files all project dependencies are defined and the generation, compilation and deployment steps are automated.

The different tools and concepts used for the implementation of NF4Droid allowed to create a sophisticated analysis tool for security experts. The intuitive web application provides rich visualisations and allows the user to thoroughly analyse the network traffic. Moreover, the server side provides solid data provisioning functionality for the client. To achieve this performance and functionality a well considered design of NF4Droid was crucial. In the next section we present this design in more detail.

6.3 Design of NF4Droid

In the following we will describe the design of NF4Droid. Since a detail description would go beyond the scope of this thesis, we only roughly provide an overview of the the important components and design approaches.

The Java and GWT based implementation of NF4Droid consists of a client and server side. For the seamless communication between the both sides we used established design concepts, supported and deployed by GWT. Furthermore, we applied common design patterns on each side for different functionalities, to build a efficient and easily maintainable application.

6.3.1 Server

The *server* side builds the core of NF4Droid. The graphic in Figure 6.2 outlines the main server components of NF4Droid, in an abstracted manner.

The server contains the *Domain Model* representing a structured view of the network traffic information, the test environment properties and the application specific informations. Furthermore, the server handles the data storage and provisioning of the information. Therefore, the *Domain Model* is saved in a relational *database (DB)*, using Hibernate for the ORM and as persistence framework (see Section 6.2.4).

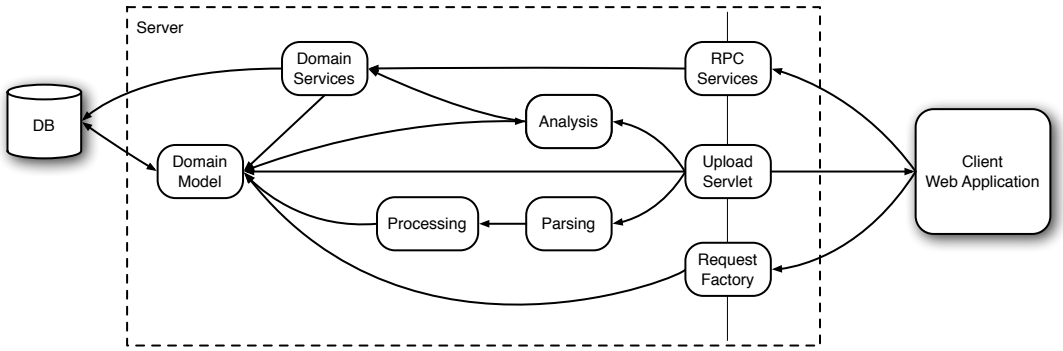

Figure 6.2: Graphic, illustrating the main components of the NF4Droid server. Moreover, it illustrates the internal and external interaction of the server components.

The information for the model is acquired from the network traffic capture files and test environment properties provided during the import by the user. The upload and import of the data is handled at the server side by the *UploadServlet* (see Section 6.2.3). Subsequently, the *Parsing*, *Processing* and *Analysis* components further process the imported data (see Sections 6.2.3, 6.2.5 and 6.2.6).

Afterwards, the information stored in the *Domain Model* is directly made accessible by the *Request-Factory* and its service proxy interfaces for the *Client Web Application* (see Section 6.2.1.1). Besides, *Domain Services* provide methods which run various queries on the *Domain Model* data, required for the data visualisation on the client side. For some complex queries the *Domain Services* even directly interact with the DB (see Section 6.2.4). Subsequently, the rehashed and aggregated data from the *Domain Services* is made available for the client, using *RPC Services* and DTOs.

We will not describe the concrete implementation of the server in all detail. However, in the next Section we outline some of the important packages and describe their functionality.

6.3.2 Package Overview

The graphic in Figure 6.3 gives an overview of the most important Java *packages* at the server side. The packages group the classes according to their functionality and represent the main components of the server. This overview should help to better understand the general structure of NF4Droid. However, for greater detail we refer to the source code of the project.

Subsequently, we describe the packages and the functionalities of the included classes:

at.tugraz.iaik.server Main package, which contains all the server related components.

at.tugraz.iaik.server.domain.model Package containing the classes representing our domain model. Additionally, the package includes AspectJ files generated from SpringRoo, which extend the basic domain classes with functionality for the persistence with Hibernate (see Section 6.2.2 and 6.2.4).

at.tugraz.iaik.server.domain.service Package including the domain service classes, providing complex query methods required for the visualisations on the client side.

at.tugraz.iaik.server.service.requestfactory RequestFactory service interface which maps the incoming requests to the according entities. Moreover, it provides the client side direct access to the domain entities over the proxy interface.

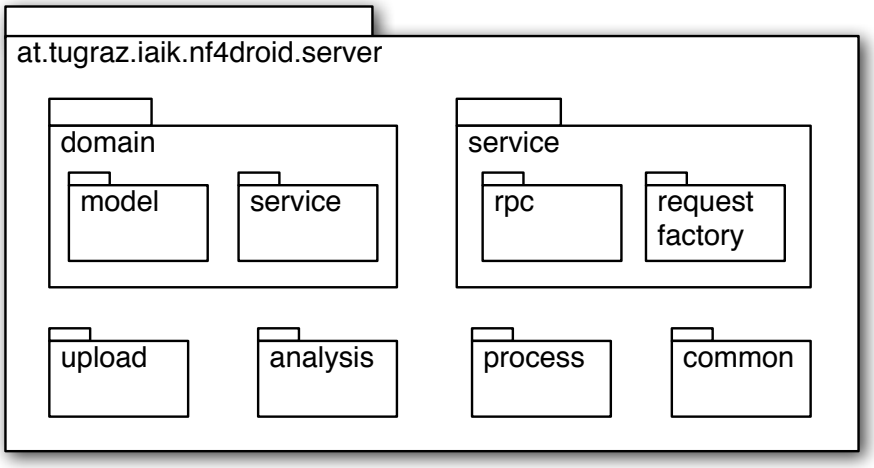

Figure 6.3: Graphic, illustrating the main Java packages and their categorisation according to their functionality at the server side of NF4Droid.

at.tugraz.iaik.server.service.rpc Classes implementing GWT-RPC services. The RPC services provide the clients indirect access to the domain service methods. However, for the data transmission we use DTOs like classes.

at.tugraz.iaik.server.upload UploadServlet class, for the handling of the traffic capture file upload and test environment properties import from the client side.

at.tugraz.iaik.server.process Package containing classes for the parsing and further processing of the uploaded network traffic capture files. Additionally, it includes classes which extend the processors and decoders classes of the Kraken library (see Section 6.2.3) for the different network protocols.

at.tugraz.iaik.server.analysis Package which includes the base class for the data exposure analysis and all the currently implemented analysis test cases. To define new data exposure tests, this is the place to start off for the developer.

at.tugraz.iaik.server.analysis Package with classes containing common functionality.

The package structure represents the logical grouping of the components at the server side of NF4Droid. The overview should help developers to understand the design of NF4Droid and assist to quickly find where certain functionality is implemented.

The server is the knowledge base of NF4Droid. It processes and brings the imported network traffic information in a format suitable for the further analysis and the fast retrieval by the client. Moreover, the server side implements the crucial analysis for the data exposure. Furthermore, it provides the data for the rich representation on the client side. The used technologies provide a solid persistence management of the data and support the communication between the client and server side using modern web technologies. Hence, the server plays a key role for the information preparation and provisioning. The client, presented in the next section, has the aim to provide comprehensive presentations and visualisations of this information.

6.3.3 Client

The client side of NF4Droid provides a web based UI for the representation and visualisation of the rehashed network traffic information form the server. The GWT based implementation uses the GWT

Activities and Places approach and a MVP like structure (see Section 6.2.1.2). The graphic in Figure 6.4 illustrates the main components at the client side and their interaction.

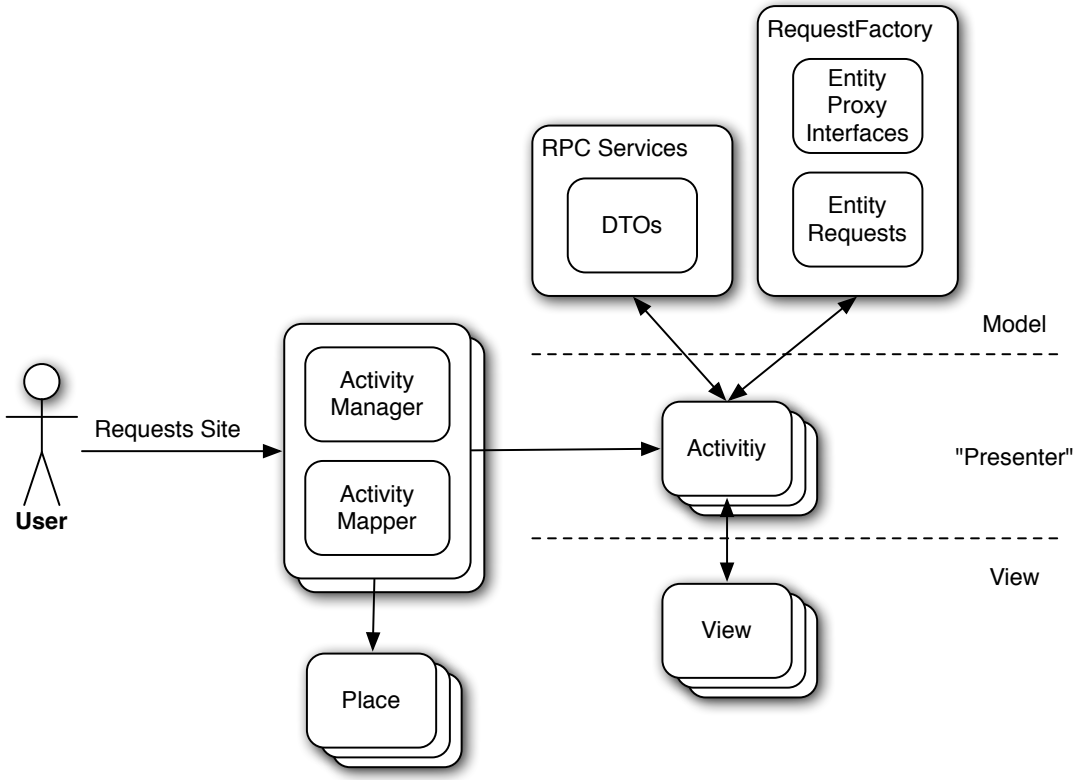

Figure 6.4: Graphic, illustrating the most important components at the client side of NF4Droid and their interaction.

If a user requests a certain site, the *ActivityManagers* associated to certain display regions makes a lookup in the *ActivityMapper*, for the *Place* associated to the requested site. Subsequently, the *ActivityManager* starts the *Activity* related to the *Place*. The *Activity* is like the presenter in the MVP concept. It communicates with the server using the the *RequestFactory* or the *GWT-RPC* services and loads the Model data for the *View*. In our case the model data is represented by the entity proxy interfaces and DTOs. Additionally, the *Activity* starts its associated *View* and handles the interaction between the Model and the View.

The implementation of the NF4Droid client side is similar to the established MVP design pattern and follows the "seperation of concerns" approach. In the next section we will describe the Java packages and their functionality at the client side.

6.3.4 Package Overview

The graphic in Figure 6.5 represents the Java packages at the client side. The packages group classes implementing certain related functionality. In the following we briefly describe the different responsibilities of the packages:

at.tugraz.iaik.client Main package, which contains the client components.

at.tugraz.iaik.client.place Package including all the Place classes, which represent a certain URLs and a corresponding state of the UI.

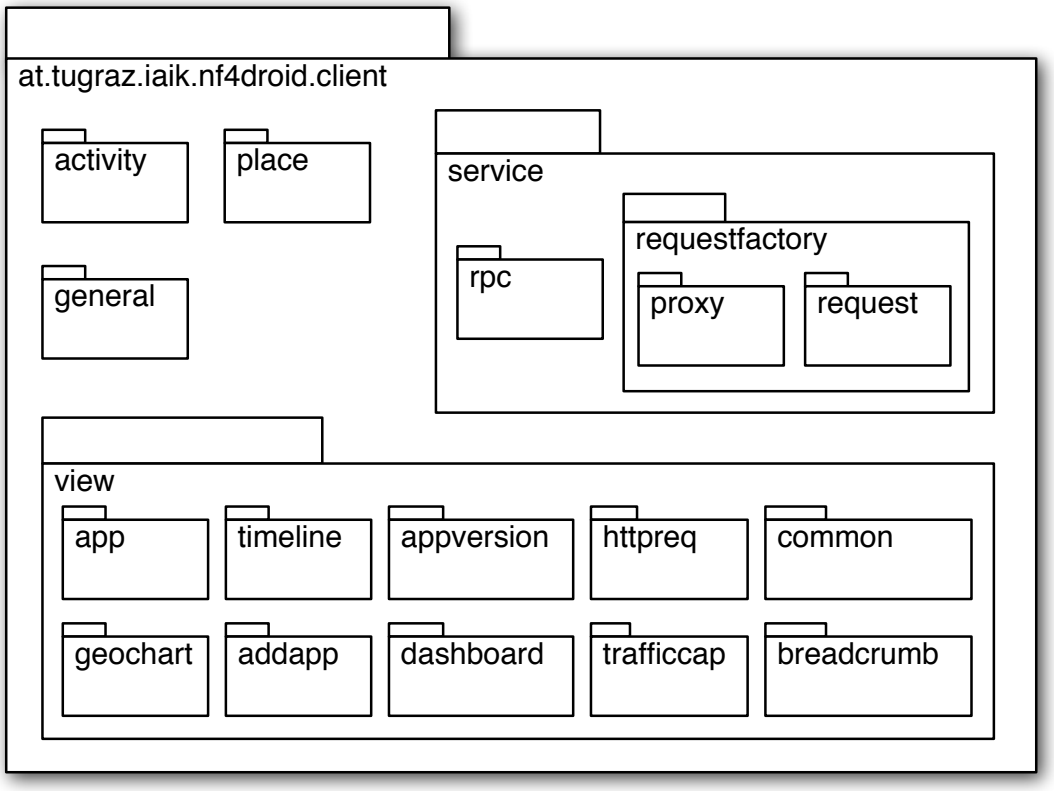

Figure 6.5: Graphic, illustrating the most important Java packages, categorised by their functionality, at the client side of NF4Droid.

at.tugraz.iaik.client.activity Package containing the Activity classes. These classes describe certain actions of the user and manage the interaction between the views and the model classes, similar to the presenter in the MVP concept.

at.tugraz.iaik.client.service.rpc Package, which includes the GWT-RPC service interfaces. The actual return types used at the service methods are DTOs, situated in the shared package.

at.tugraz.iaik.client.service.requestfactory Package containing the RequestFactory and the related entity proxy and request interfaces. This classes (together with the DTOs) present the model part of the MVP concept.

at.tugraz.iaik.client.service.views Package, which contains the different view interfaces, classes and UiBinder files of the UI. This files represent the view part of the MVP concept.

at.tugraz.iaik.client.service.views.app Package for the applications list view.

at.tugraz.iaik.client.service.views.appversion Package for the application versions list view.

at.tugraz.iaik.client.service.views.trafficcap Package for the traffic captures list view.

at.tugraz.iaik.client.service.views.dashboard Package for the dashboard view.

at.tugraz.iaik.client.service.views.geochart Package for the traffic geo chart view.

at.tugraz.iaik.client.service.views.timeline Package for the traffic timeline view.

at.tugraz.iaik.client.service.views.httpreq Package for HTTP requests list view.

at.tugraz.iaik.client.service.views.breadcrumb Package for the breadcrumb part of the UI.

at.tugraz.iaik.client.service.views.common Package containing common view classes and general UI components.

at.tugraz.iaik.client.general Package, which includes general classes for the UI like for the the setup of the communication between the client and server or for the history management.

The client side has to aim to provide the user a rich presentation of the visualisation of the data provided by the server. With the concepts provided by GWT and the use of general design patterns, NF4Droid implements a well structured design.

6.3.5 Shared

Beside the server and client package, we have a package (at.tugraz.iaik.shared) containing classes which are used by both sides. It contains DTOs used by GWT-RPC and some constant enum types.

We tried to design NF4Droid in a way that it follows established design patterns and is easily maintainable. The various tools we deployed during the development like GWT or SpringRoo supported as in this task. The use of modern web technologies allowed us to build a interactive and intuitive network traffic analysis tool for security experts.

6.4 Deployed Technologies and Tools

In this section we enumerate and briefly describe all the technologies and tools used for the development and implementation of NF4Droid.

6.4.1 Spring Roo

Spring Roo [174] is rapid application development tool for Java. It allows developers to fast and easily create applications and the corresponding object models, compliant with established design patterns. Therefore, Spring Roo integrates and supports various Java technologies like the JPA, the Spring Framework, Hibernate and Google Web Toolkit. Spring Roo is licensed under the Apache License version 2.0 [87].

6.4.2 Google Web Toolkit (GWT)

Google Web Toolkit (GWT) [72] is open source development toolkit for the creation of front-end web applications and the corresponding server back-ends. The Java APIs and widgets provided with the GWT SDK allow to write AJAX based applications in Java. The Java code is subsequently compiled to highly optimized JavaScript that runs across all browsers. This level of abstraction, on top of common AJAX related data manipulation and communication concepts, allows to easily build sophisticated browser-based applications. Hence, developers can design and develop their applications in a pure object-oriented fashion, with Java as high-level programming language. GWT provides solutions for tasks like asynchronous remote procedure calls (RCPs), history and bookmark management, internationalisation, localisation and cross-browser compatibility. Moreover, include libraries of the GWT SDK dynamic and reusable UI components, which allow to quickly build rich UIs. GWT is licensed under the Apache License version 2.0 [87].

6.4.3 Hibernate

Hibernate [113] is a open source persistence framework, which includes functionality for the ORM of Java object-oriented domain models to traditional relational databases. It generates the data definition language (DDL) scripts for the creation of the database schema, according to the domain model and the defined mapping. The mapping can be defined by the developer using XML configuration files or Java annotations. Furthermore, Hibernate offers high-level object handling functionalities, which facilitate and automate the create, read, update and delete (CRUD) operations. Additionally, it supports various relational database management systems (RDBMSs) and provides connection, session and transaction management. Hibernate is licensed under the GNU Library General Public License version 2.1 [86].

6.4.4 Java Persistence API (JPA)

The Java Persistence API (JPA) [151] is a framework providing standardised interfaces and implementations for the ORM of entity classes and the creation of criteria queries. JPA is supported by multiple persistence frameworks like Hibernate and, accordingly, it improves the exchangeability of the underlying persistence framework.

6.4.5 QueryDSL

QueryDSL [160] is a open source framework for the creation of type-safe SQL-like queries using a Java API. Hence, domain types and properties can be referenced safely and syntactically invalid queries are avoided. Moreover, it supports various Java relational data management back ends including the JPA. QueryDSL is licensed under the Apache License version 2.0 [87]

6.4.6 GWTUpload

GWTUpload [144] is a open source Java library, which allows to provide advanced file uploads with progress bars in GWT. It consists of a servlet at the server side and a client side component made with GWT. The two components communicate using AJAX to exchange the current upload status. GWTUpload is licensed under the Apache License version 2.0 [87].

6.4.7 Kraken

Kraken [129] is a open source Java information security suite. The network forensic part of the suite offers a Java based PCAP parser, a TCP/IP stack implementation and various decoders for application layer protocols. In contrast to most libraries for the processing of PCAP files, it is purely Java based. Kraken is licensed under the Apache License version 2.0 [87].

6.4.8 jnr-netdb

jnr-netdb [138] is a open source library, which provides a Java API for the mapping between TCP and UDP service ports and corresponding service names. It tries to use the native system functions `getservbyname` and `getservbyport`. Additionally, it provides fallback methods, which parse the `/etc/services` system file or use a inbuilt mapping table. jnr-netdb is licensed under the Apache License version 2.0 [87].

6.4.9 MaxMind IP Database

“GeoLite Country” and “GeoLite City” are freely available IP geolocation databases, offered by the MaxMind company [117]. The databases allow to approximately map IP address to locations. Updated

versions of the database are published every month. The databases are licensed under Creative Commons Attribution-ShareAlike 3.0 License [56].

Additionally, a Java API, for the access of the information stored in the database files, is available [135]. The API is licensed under the GNU Library General Public License version 2.0 [85].

6.4.10 Twitter Bootstrap

Bootstrap [186] is a open source project from Twitter, with the aim to provide a powerful front-end framework for the fast and easy web development. It allows to build intuitive and great looking web UIs, by supplying grid layouts, dozens of components, form controls and plugins. Twitter Boostrap is licensed under the Apache License version 2.0 [87].

6.4.11 GWT-Bootstrap

GWT-Bootstrap [111] is a open source toolkit, which allows to directly use the Twitter Bootstrap interface library in GWT. It provides a Java API for the access and use of the UI components provided from Bootstrap. GWT-Bootstrap is licensed under the Apache License version 2.0 [87].

6.4.12 Highcharts JS

Highcharts JS [41] is a JavaScript charting library, which allows to easily create great looking interactive charts. It supports various chart types like pie, line, area, bar and many more, which are rendered using HTML5/Scalable Vector Graphics (SVG)/Vector Markup Language (VML) technologies. Highcharts JS is free for non-commercial use and licensed under the Creative Commons Attribution-NonCommercial 3.0 license [55].

6.4.13 GWT Highcharts

GWT Highcharts [110] is a open source toolkit, which allows to directly use the Highcharts JS visualisations library in GWT. It offers a Java API for the use of the JavaScript based visualisations of Highcharts JS.

6.4.14 Google Chart Tools

Google Chart Tools [71] is a free library for the creation of well-designed interactive charts. The different chart types like pie, line, bar or geo can be created using a JavaScript API and are rendered using HTML5/SVG/VML technologies. To use the Google Chart Tools library you have to accept the Google APIs Terms of Service [105]. The Chart Tools API Library for GWT [107] allows to directly use the Google Chart Tools in GWT with a Java API.

6.4.15 AspectJ

AspectJ [89] extends Java with Aspect-Oriented Programming (AOP) features. The AOP features can be used to increase the modularity of the software and to provide separation of cross-cutting concerns. Use cases are error handling and checking, context-sensitive behaviour or monitoring and logging. AspectJ is licensed under the Eclipse Public License 1.0 [90].

6.4.16 Maven

Apache Maven [88] is a software project management and comprehension tool. It provides functionality to manage and automate project building, reporting and documentation. Additionally, it offers features for the management of internal and external dependencies. XML based POM files allow the define the complete configuration of the project, including things like the build process or the project dependencies. Maven is licensed under the Apache License version 2.0 [87].

All the deployed tools did support the development process and allowed to build NF4Droid with all its features and functionalities. Technologies like GWT offer comprehensive functionality for the creation of modern web applications. Tools like SpringRoo allow to quickly create projects following established design patterns. Moreover, only the use of the various libraries for the creation of the UI and the visualisations enabled us to build the great looking and informative network traffic analysis tool NF4Droid.

However, the incomplete documentation of some projects and the use of various frameworks, new for us, made it sometimes hard to focus on the real task, the analysis of the network traffic from Android applications.

Chapter 7

Case Study and Results

To analyse the behaviour of applications and to test the capabilities of NF4Droid, we examined popular applications and known malware in a case study. In the first part of the study we will examine the top 50 free applications from Google Play, since they are commonly installed and used by a large user base. In the second part we analyse known malware, to test the malware detection capabilities of NF4Droid.

Before we present our analysis results, we describe the deployed testing method in the next section.

7.1 Testing Method

As test device for the analysis of the applications we used a rooted ZTE Blade [60] smartphone running CyanogenMod-7.1.0-Blade [63] with Android version 2.3.7. The device was connected with the Internet using the Wi-Fi, and a SIM card of the Austrian network provider Tele.Ring [180] was inserted. Subsequently, we manually and separately tested every application.

For the capturing we used the tcpdump-tool (see Section 5.1.1.1), which we instructed with the Android Debug Bridge (adb) [69] command line tool and shell scripts. Similarly, we automated the installation and uninstallation of the APK files and the launch and stop of the applications. Therefore, user interaction was only required to generate user input for the application.

Every application was executed for at least 30 seconds and provided with user input, to simulate normal application use. However, due to the amount of analysed applications we only tested the rudimentary functions, what might leave some parts of the application unexposed.

To provide a realistic test environment and to be able to run further analysis, we prepared the system with some sample user data like user accounts, contacts, SMS, mails and calendar entries. Besides, we logged all the test environment properties, required for the in-depth analysis with NF4Droid.

Subsequently, we imported the network traffic captures of the applications together with the test environment properties to the NF4Droid platform, where the further processing took place. Afterwards, NF4Droid was deployed to thoroughly examine the gathered results and to understand and inspect the application behaviour. The according results are presented in the next section.

7.2 Top 50 Free Applications

The Table 7.1 presents the top 50 free applications at Google Play on the 29^{th} of June 2012, which we analysed in the first part of our case study. The exact ranking algorithm of Google Play is not revealed, however, it is likely based on information like user ratings, total downloads and install/uninstall rates [51]. Accordingly, the applications are a representative test base for our analysis, since they are commonly used by many users. Some of the applications have been omitted and replaced with subsequent

ones in the ranking, since they were incompatible with the test device (ZTE Blade [60]) or network provider (Tele.Ring [180]) we used.

We deployed the testing method, described in the previous section, to thoroughly analyse the applications for various aspects. Thereby, we looked into general aspects like the total traffic amount, the deployed protocols and the locations of the contacted servers. Furthermore, we examined the results of the in-depth analysis for information exposure. Additionally, since previous work by The Wall Street Journal [183] and Fulton [92] already indicated, that some applications commonly share sensitive information for advertising purposes, we especially investigated this issue.

7.2.1 Results

The Table 7.1 lists the complete "application name" and "application version" information from the tested applications and presents the detailed results of our analysis. The column HTTPS denotes, if an application uses the secure communication protocol to some extent. Furthermore, the columns location, Android ID, IMEI and IMSI indicate, how often certain information is exposed by an application in the captured network traffic.

In the following, we briefly describe some general aspects about the applications, which we identified during the analysis with NF4Droid.

7.2.1.1 General

The general analysis reveals, that *42 (84%) out of the 50 tested applications use secure communication (HTTPS)*. However, from the overall TCP and UDP traffic *only an average amount of 19.6% per application is HTTPS traffic*. Whereas the *majority of the traffic is HTTP with an average off 76.5%*. Additionally, we identified, that applications developed by bigger companies like Google, Facebook or Dropbox more likely, and with a greater share in the overall network traffic, deploy secure communication technologies.

Besides the analysis of the general network traffic information, we evaluated the results from the in-depth analysis, presented in the next section.

7.2.1.2 Information Exposure

Our investigation for exposure of sensitive information uncovered the following "alarming" facts. As illustrated in the Figure 7.1, at least *29 (58%) of the 50 tested applications expose information* in the HTTP network traffic. From all the tested applications *29% exposing the Android ID*, *15.1% the IMEI*, *12.6% the location* and *1.3% the IMSI*. Furthermore, *13 (26%) of the 50 applications expose information of more than one type*.

However, for *none of the applications we detected the exposure of the phone number, the user name, the password, the SSID or the BSSID*. At this point, we have to mention, that information, which is exposed over secure communication channels, is not detectable with the deployed analysis and capturing method (see Section 5.1). Furthermore, since the analysis method of NF4Droid has certain limitations (see Section 5.2.3.4), applications might expose even more information, than we could find with our analysis.

To gain more knowledge about the reasons, for what applications expose certain information, we examined the results in greater detail. This led to the results, presented subsequently.

I. Advertising The further inspection of the results led to the discovery that *26 (89.7%) of the 29 applications*, which expose information, *share the information solely for advertising purposes*. Only the

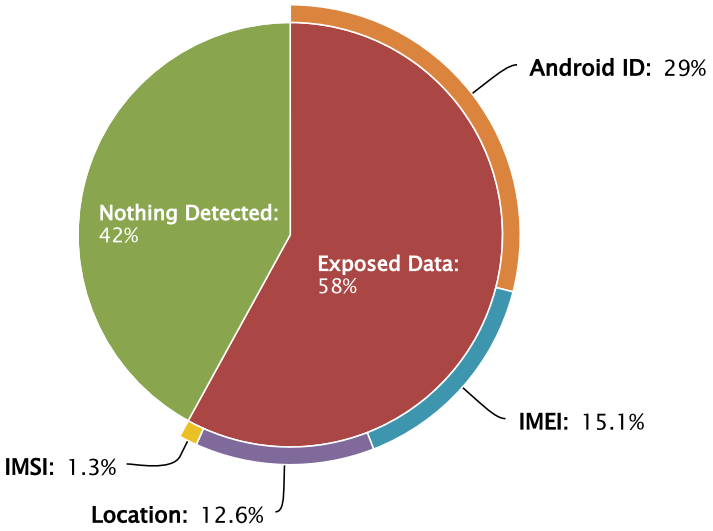

Figure 7.1: Chart, illustrating the number of the top 50 free applications, exposing information in the network traffic. The inner circle indicates the percentage of applications, where we detected exposure of information. The outer circle around the exposed data denotes the ratio between the different exposure types.

applications "Google Voice Search", "Google Search" and "Barcode Scanner" use the detected information for desired purposes, required for the full functionality of the application. These findings are in line with the research results from The Wall Street Journal [183] and Fulton [92], which as well identified, that applications commonly share information with advertisers.

The chart in Figure 7.2 enumerates the domain names of all advertising providers, to which the tested applications send information. Furthermore, it shows the number of applications, which used a certain service, where some of the applications used more than one advertising provider.

This results show that developers of free Android applications heavily use advertising to finance their development and earn money. However, the privacy of the user is threatened if various information is shared with advertising companies. Advertisers use the information to gain detailed knowledge about users and to track their behaviour, as described in more detail in Section 2.1.3. To emphasise our findings, we present in the following some concrete examples.

II. Selected Examples In the following we will examine some examples in greater detail, to reinforce our general results, presented before. The selected applications are popular Android applications, which however show heavy advertising habits or unexpected behaviour.

Magic Piano is a virtual piano application/game, downloaded more than 5 million times. However, the free version of the application integrates three different advertising providers (`mydas.mobi`, `mopub.com` and `doubleclick.net`), to which the Android ID, the IMSI and the location is exposed. An interesting fact is, that the application requires the permission to access the coarse location. The permissions is used in a sharing feature, where users can share their songs with others. In spite of that, the application even uses the permission to share the location information with advertising companies. At least, the application developers provide a detailed privacy policy, where the use of the information for advertising purposes is denoted [173].

Cut the Rope is a very popular game, which is even advertised on the official Android website [98] (see Screenshot 7.3). The free version of the brain and puzzle game was downloaded more than 1 mil-

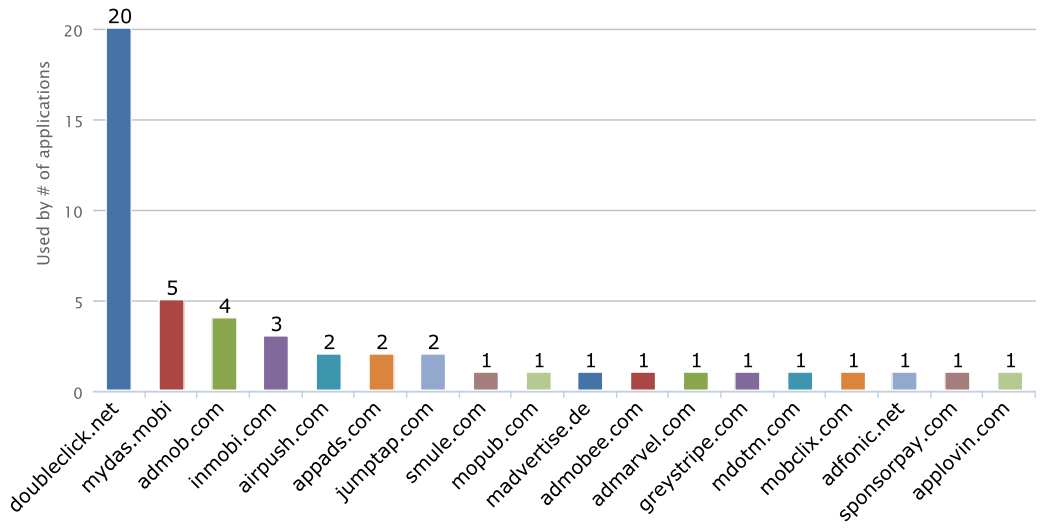

Figure 7.2: Chart, illustrating the advertising providers, to which the tested top 50 free applications exposed information. Additionally, the number of applications, which used a certain service, are denoted.

lion times. Our analysis, however, reveals that the game exposes the Android ID, IMEI, IMSI and location to five different advertising providers (`greystripe.com`, `jumptap.com`, `inmobi.com` and `doubleclick.net`). With this result it is the application, exposing the most information in our test. Additionally, the manual inspection showed, that the game exposes even more information like the size of the external memory. Nevertheless, the developers provide a privacy policy, where they denote, that they collect and share none-personal information [196]. The Screenshot 1.1, presented in the introduction chapter, shows a part of our analysis results for this application.

wetter.com is a weather forecast application, downloaded more than 5 million times. It is understandable, that the application requires the permission to access the location to provide local weather information. Nevertheless, the application exposes the location information to an advertiser (`amobee.com`). Accordingly, this is an example for the use of a permission, and the according sensitive information in a possible unexpected way for the user. Furthermore, this advertising habits are not directly denoted in the imprint. However, the developers state that the track information for statistical analysis (see imprint in the application).

Although some applications have EULAs or privacy policies, where they declare their behaviour, many applications do not provide any information about their advertising habits. Furthermore, the question remains, if users read the policies and are able to understand the conditions and their consequences. Additionally, some developers claim, that they only expose non-personal information by sharing only unique identifiers. However, if advertisers get more detailed information from other applications, they are able to correlate it to the specific unique identifier, exposed from one application.

The selected examples should highlight some of our findings and help to understand the results of the study. In the following we give a summary about the study results.

7.2.1.3 Conclusion

The results show, that the majority of the tested top 50 free applications expose information. The information is in almost 90% of the cases, transmitted solely for advertising purposes. Although, most of the applications “only” expose unique identifiers, some already reveal more sensitive information like the

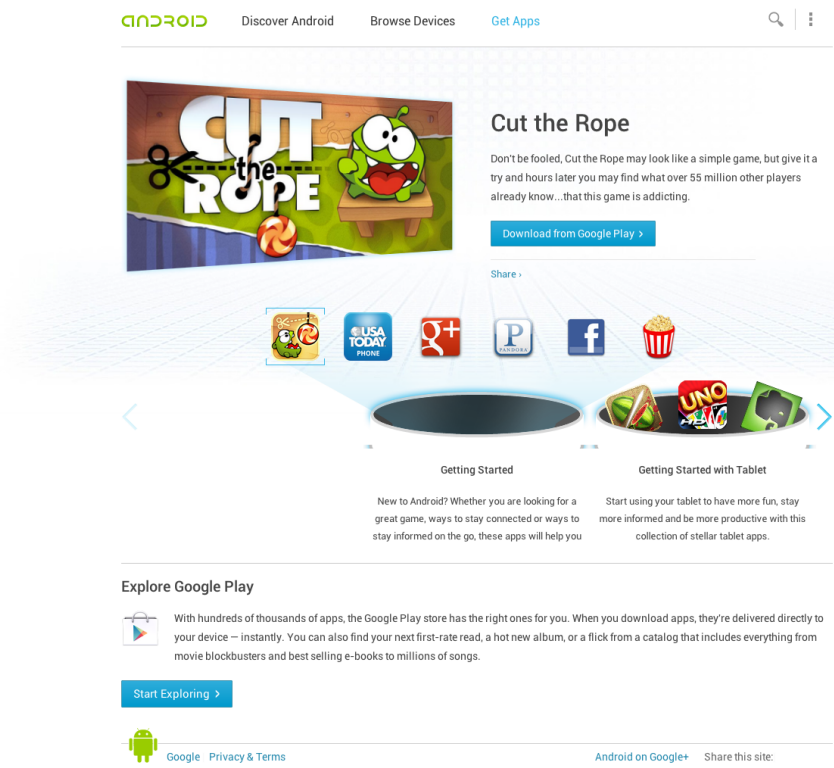

Figure 7.3: Screenshot, showing the advertisement for the Cut the Rope application on the Android website [98].

location. Such advertising behaviour of applications is, however, hard to identify for the user. Only some applications provide privacy policies or EULAs, denoting the advertising. In most cases the users might never notice the exposure of the sensitive information.

It is worth to mention, that many very popular applications like “Cut the Rope free”, which are even advertised by Google (see Screenshot 7.3), heavily integrate advertising in their applications. In our study, the most popular applications were those, which expose the most information with a large number of advertisers.

Furthermore, the study reveals, that applications use their required permissions not only for their offered functionality, but also for advertising purposes. This makes it even harder for the user to judge, if certain permissions are legitimate for an application.

A positive fact is, that we did not find an application in the test, which exposed highly sensitive information like the phone number or user passwords. However, since our analysis method has certain limitations (see Section 5.2.3.4), the probability exists, that we just did not detect further exposure.

Another observation is, that in the top 50 free applications 8 applications are developed by Google, which do not expose information in a undesired way. Accordingly, the statistic might have been even worse, if we would have omitted the applications from Google in the ranking.

Relating to NF4Droid we can conclude, that it provides great functionality for the analysis of the network traffic. It was especially useful for the analysis of the general behaviour of the applications and allowed to thoroughly inspect their advertising habits. In the following section we will test the analysis capabilities of NF4Droid for malicious applications.

#	App Name	Package	App version Name	App version Code	HTTPS	Location	Android ID	IMEI	IMSI
1	Street View on Google Maps	com.google.android.street	1.8.1.0	18100	-				
2	WhatsApp Messenger	com.whatsapp	2.7.973	29736	x				
3	Gmail	com.google.android.gm	2.3.6	176	x				
4	Facebook for Android	com.facebook.katana	1.9.6	21878	x				
5	Maps	com.google.android.apps.maps	6.9.0	609001402	x				
6	Logo quiz	logos.quiz.companies.game	4.5	25	x		26		
7	YouTube	com.google.android.youtube	2.4.4	2404	x				
9	Voice Search	com.google.android.voicesearch	2.1.4	214	x	4			
10	100 Floors	com.tobiapps.android_100fl	2.2.0	22018	-				
11	Skyjumper	com.neonnighthawk.base.android	2.0.0	3	x				
12	Penalty ShootOut football	com.box10.penaltyshootout	1.0.1	2	x	6			
13	Google Search	com.google.android.googlequicksearch	1.3.3.247963	133247963	-	3			
14	Google+	com.google.android.apps.plus	3.0.0.31440702	300314407	x				
15	Official UEFA EURO 11 app	com.imano.euro2012	2.2	6	x				
16	War of Repdroduction	org.appplus.tadpolewarGoogle	1.95	15	x				
17	1001 Beleidigungen HD+ free	com.RiSaSolutions.beleidigungen	1.2	3	-		3	9	
19	Dooors - room escape game	com.kuhakuworks.DOOORS	2.0.3	5	x		18		
20	Skype - free IM & video calls	com.skype.raider	2.8.0.920	34079640	x				
21	Magic Piano	com.smule.magicpiano	1.0.2	102	x	5	14	12	
22	Car Logos Quiz	com.waylandindustries.carlogos	1.3	13	x		17		
23	Logo Quiz	de.androidcrowd.logoquiz	1.6	7	x		18	4	
25	Dropbox	com.dropbox.android	2.1.5	21500	x				
26	Logo Quiz - World Flags	flag.quiz.world.national.names.learn	2.2	13	x		12		
27	Tiny Flashlight and LED	com.devuni.flashlight	4.9.1	135	x		13		
28	Puzzles with Matches	com.celticspear.matches	1.5.6	21	x		2		
29	wetter.com	com.wetter.androidclient	1.3.9	18	x	31			
30	Bow Man	com.shootinggames.bowman	1.1	3	x		4	9	
31	Adobe Reader	com.adobe.reader	10.2.1	60548	x				
32	Shazam	com.shazam.android	3.9.3-BB75008	75008	x	25	24		
33	Angry Birds	com.rovio.angrybirds	2.1.1	2110	-		6	7	
34	100 Floors Official Cheats	com.tobiapps.android_100floors_guide	2.0.0	20099	x				

#	App Name	Package	App version Name	App version Code	HTTPS	Location	Android ID	IMEI	IMSI
35	Heute - Die Tageszeitung	at.heute.android	1.0.2	102	x				
36	Simple mp3 downloader	org.goldennuggetapps.simpledl	1.1.8	26	x	2			
37	Cut the Rope Free	com.zeptolab.ctr.lite.google	1	1	x	10	66	81	16
38	GO Launcher EX	com.gau.go.launcherex	3.05	112	x				
40	Facebook Messenger	com.facebook.orca	1.8.002	24034	x				
41	Garfield's Diner	com.webprancer.google.GarfieldsDiner	1.1.2	7	x		5	4	
42	How to Tie a Tie	com.artelplus.howtotie	2.2	22	-		5		
43	Fruit Ninja Free	com.halfbrick.fruitninjafree	1.6.2.10	1603	x		10	7	
44	Trial Xtreme 2 Winter	com.galapagossoft.trialx2_winter	2.15	152	x		37	45	
46	Free Kick Euro	com.xupeaceful.freekickeuro	1.0.0	1	x		5		
47	Bubble Shoot	om.game.BubbleShoot	1.3	4	x		7		
48	SCOTTY mobil	de.hafas.android.oebb	2.2.4	202041	x				
49	Barcode Scanner	com.google.zxing.client.android	4.2	84	-	2		6	
50	Flipboard: Your News Magazin	flipboard.app	1.8.4	105	x				
51	Angry Birds Space	com.rovio.angrybirdsspace.ads	1.2.1	1210	x	1	8	24	
53	Google Translate	com.google.android.apps.translate	2.4.2	103	-				
56	PicsArt Photo Studio	com.picsart.studio	3.0.6	44	x		1		
57	MyCalendar Free	com.kfactormedia.mycalendarmobile	2.52	63	x		2	6	
58	Scare Your Friends SHOCK!	scare.your.friends.prank.maze.halloween	5.3	8	x		4		
				Total:		89	307	214	16
				Number of apps:	42	9	23	12	1
				Percent from all apps:	84%	20%	46%	24%	2%

Table 7.1: Table of the top 50 free applications at Google Play on the 29^{th} of June 2012 analysed with NF4Droid.The column HTTPS indicates if an application used, to some extent, secure communication. The columns Location, Android ID, IMEI and IMSI denote how many times an application exposed a certain information. Applications incompatible with the test device (ZTE Blade) or the service provider (tele.ring) have been omitted and replaced with subsequent applications in the ranking.

7.3 Known Malicious Applications

For this part of our case study, we analysed known malicious applications with NF4Droid. The Table 7.2 lists the different types of malware and the according applications, we analysed with the test method, described in Section 7.1. The different types of malware have been selected, based on the description, given at the "forensic blog" [83] and their according probability to expose information over the network.

The malicious applications for our analysis where mainly provided from the Android Malware Genome Project [198]. Additionally, some malware samples where downloaded from "contagio mobile" website [58].

During the analysis, we looked into general aspects of the network traffic and tried to identify the exposure of information with the in-depth analysis capabilities of NF4Droid. The results are presented in the following section.

7.3.1 Results

The Table 7.2 enumerates the 19 different types of malware, from which we selected applications and tried to analyse the network traffic. However, at 6 out of the 19 categories the selected application did not produce any network traffic. To confirm, that this type of malware does really not produce any network traffic, we tried two more applications of the same category, leading to the same result. Accordingly, for 6 types of malware we could not analyse the network traffic and the categories are in the following omitted.

The Table 7.2, additionally presents the detailed results from our analysis. Similarly to our previous case study the column HTTPS denotes, if an application uses the secure communication protocol to some extent. The columns location, Android ID, IMEI, IMSI and phone number show, how often certain information is exposed by an application.

To identify the malicious behaviour of the tested applications, we first evaluated general aspects, presented in the next section.

7.3.1.1 General

The general analysis shows that *only 3 (23.1%) of the 13 malicious applications use secure communication (HTTPS)*. Moreover, the majority of the TCP and UDP traffic per application is HTTP traffic with an average of 84.9%. This results highlight, that most of the analysed malicious applications use conventional network protocols and do not deploy secure communication technologies.

Although the general analysis enabled us to get a first insight about the network traffic, produced by malicious applications, it did not allow us to directly identify malicious behaviour. Further knowledge about the malware and thorough manual inspection would be required. Hence, we examined the results of the automated in-depth analysis from NF4Droid, presented in the following section.

7.3.1.2 Information Exposure

To determine the malicious behaviour of our tested applications, we evaluated the results of the in-depth analysis. As presented in the Figure 7.4, we detected that *10 (76.9%) of the 13 analysed applications exposed information* in the network traffic. From all tested applications, *25.6% expose the IMEI*, *18.3% the Android ID*, *14.7% the location*, *11% the phone number* and *7.3% the IMSI*. Still, for *none of the applications we detected the exposure of the user name, the password, the SSID or the BSSID*, with the deployed analysis.

The further investigation about, to whom the malicious applications send the information, shows that, 7 (70%) of the 10 applications expose information to advertising companies. The remaining 3 (30%) of

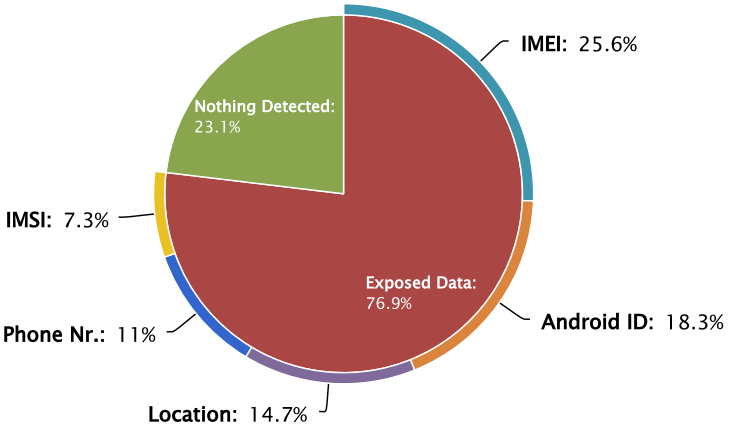

Figure 7.4: Chart, illustrating the number of tested malicious applications, exposing information in the network traffic. The inner circle indicates the percentage of applications, where we detected exposure of information. The outer circle around the exposed data denotes the ratio between the different exposure types.

the 10 send the information to websites, which seem to have a legal purpose. Accordingly, we could not identify direct malicious behaviour at any of the tested applications. Still, we were able to reveal, that even malware integrates advertising.

The fact, that malware exposes information to advertisers, raises even more concerns about the privacy of the user, since 3 of the malicious applications share the phone number with advertising companies. Hence, advertisers can match the unique identifiers, sent together with the phone number, with the unique identifiers sent from "normal" applications. This allows advertisers to aggregate more and more information specific for a certain device and person. Accordingly, it makes the non-personal information, collected from "normal" applications to some degree personal.

Although the in-depth analysis could not reveal direct malicious behaviour, it revealed interesting facts about malware, summarized in the next section.

7.3.1.3 Conclusion

The analysis of the 19 different types of known malware in this part of the case study led to following results.

The tested malware mainly (84.9%) uses HTTP for the communication over the network and only 12% of the traffic per application where secured (HTTPS). This reinforces the possibility to detect malicious behaviour in the network traffic. Nevertheless, the risk remains, that malware transmits sensitive information, using secure communication.

The analysis for exposure of information was not able to identify the direct malicious behaviour of applications. However, it revealed that 7 (70%) out of 10 malicious applications integrate advertising. Malware exposes not only unique identifiers to advertisers, but also data like the phone number and the location. The information, gathered by advertisers from malicious applications together with information from "normal" applications, allows to already present a quite personal profile of individuals. Accordingly, this concerns the privacy of the user.

NF4Droid was not able to directly reveal malicious behaviour. However, it allowed to get insights about the the behaviour of malware. Moreover, the general analysis capabilities could be used to manually inspect the network traffic of malicious applications. Nevertheless, this would be a time-consuming process and detailed knowledge would be required.

Our overall case study presents interesting results from the network traffic analysed for different types of applications. With a test base, consisting of popular free applications and known malware, we could test different kind of applications, which led to the same result, that applications commonly integrate advertising in their applications. NF4Droid was able to reveal this advertising habits and allowed to investigate general aspects of the network traffic. However, NF4Droid could not directly reveal malicious behaviour. Still, NF4Droid provides a solid platform for the thorough analysis of the network traffic from Android applications.

Malware Type	App package	App version Name	App version Code	HTTPS	Location	Android ID	IMEI	IMSI	Phone number
PJAPPS	org.jiaxxhaha.netraffic	1.1.1	45			1			2
AnserverBot	com.keji.danti625	3.0.8	32						
Basebridge	com.keji.danti207	2.4	15				3		
BgServ	com.virsir.android.chinamobile10086	2.2	14		2		5		
DroidDreamLight	cn.com.opda.android.super.clearmaster	2.0.1	112			1	7		
Droid Kung Fu	com.sansec	V1.0.09	9		4	6			6
GingerMaster	com.igamepower.appmaster	1.0	1004	x			17	17	17
GoneIn60Seconds	com.gone60	1.09	15						
KMIN	km.home	1.0	1				1	1	
Plankton	com.crazyapps.angry.birds.multi.user	1.00	1				35		
Sndapps	com.typ3studios.mosquito	1.4	8						
Geinimi	com.handcn.GoldMiner.free	1.2	3	x	2	6			
GoldDream	ca.rivalstudios.runboyrun	1.2.1	4	x	6	5	21		
Jsmshider	net.sunyidingophone.environmental	1.0	1	Tested, but produced no network traffic. Omitted in the overall result.					
Spitmo	org.android.system	1.0	1						
Nickispy	com.nicky.lyyws.xmall	1.0	1						
TapSnake	net.maxicom.android.snake	1.2.2	122						
YZHC	hamstersuper.client.game	1.0	1						
Crusewind	com.flashp	101	1						
			Total:		14	19	89	18	25
			Number of apps:	3	4	5	7	2	3
			Percent from all apps:	23.1	30.8%	38.5%	53.8%	15.4%	23.13%

Table 7.2: Table enumerating the 19 different types of malware and the according applications, which we tested. The column HTTPS indicates if the malware used, to some extent, secure communication. The columns Location, Android ID, IMEI, IMSI and Phone number denote how many times an application exposed a certain information. 6 of the 19 tested malicious applications we tested did not produce any network traffic. Accordingly, the are omitted in the overall result.

Chapter 8

Conclusion and Outlook

In this chapter we will summarise the general results of the thesis and outline some future work.

8.1 Summary

The aim of this thesis was the development of a *network traffic analysis* method for Android applications. The analysis method should reveal *general information* about the network traffic like how much data was transmitted to which servers and what protocols were deployed by the applications. Beside the general aspects, we wanted to be able to *identify the exposure of sensitive information* through detailed examination of the network traffic. All this analysis result should help security experts to understand the functionality of applications and to *unveil possible unwanted or even malicious behaviour*.

We started with the identification of the *threat types* for mobile platforms and categorised them into *personal spyware, malware and grayware* according to their behaviour and objective. Moreover, we highlighted the different *motives for the development* of malicious applications, where the *gathering of user information* was one of the most important facts relating to the network traffic analysis. Since mobile platforms provide *security measures* which should prevent the development of malicious applications and lower the possible impact, we gave a short description of the different measures and pointed out their limitations. Aspects like the openness and the high market share made the Android platform to *one of the main targets* for the development of malicious applications. Accordingly, our work focuses on the analysis of the network traffic captured from Android applications. Additionally, the Android platform offers more technical possibilities for the capturing of the network traffic.

Different *application analysis and malware detection methods* have been developed by researchers, which should help to identify malicious applications. The methods follow different approaches and vary in their requirements on the *analysis environment*. Methods like the *static analysis* require almost no interaction of the user and are fast, but however reverse engineering technologies need to be applied to reconstruct the source code. In contrast, the *dynamic analysis* requires the time consuming execution of the application and the interaction of a human or the simulation of the user input. However, during the direct execution, the applications show their real behaviour. High-level methods like the *market metadata analysis* even do not require the application package, but are only able to provide general results. Accordingly, no analysis methods is superior to any other. Each of them has different advantages and disadvantages in terms of complexity, the requirement of human interaction and the comprehension of the results.

The examination of existing *network traffic analysis* methods revealed that only less research has been made on the analysis of the network traffic of Android applications. Since the network is the main interface to outerworld of today's mobile devices, it has great potential to reveal the behaviour of applications. However, security experts just used standard network traffic analysis tool, which were not

specialised for the use case of the Android application network traffic analysis. Hence, we pointed out that a specialised network traffic analysis method could not only reveal general communication aspects of applications, it might further be able to identify the exposure of sensitive information.

Motivated by this results, we developed the *network traffic analysis tool NF4Droid*. NF4Droid is specialised for the analysis of the network traffic captured from Android applications. It provides various *presentations and visualisations* of the network traffic information, which should allow security experts to understand the behaviour of applications. Moreover, it offers *in-depth analysis features* for the identification of information exposure. NF4Droid is implemented as desktop web application an developed for security experts. Although the capturing of the network traffic is not part of NF4Droid, we describe possible capturing methods. The implemented automatic analysis features follow the simple approach, to search in the network traffic for occurrences of the test environment properties provided by the user, together with network traffic capture file. Additionally, to achieve better results during the identification of data exposure, we imitate obfuscation methods deployed by applications, like hashing.

To examine the network traffic of Android applications and to evaluate the capabilities of NF4Droid we analysed the *top 50 free applications from Google Play* and some *known malicious* applications. The analysis yield to *general results* like how much data was transmitted by the applications, what protocols where deployed or which servers have been contacted. Additionally, the *in-depth analysis* of the traffic identified the *exposure of information*. For the free applications we found out that 84% of the applications use secure communication, but still the average ratio of the secured traffic of the overall TCP and UDP traffic per application is less than 20%. Hence, HTTP is still the protocol used the most by the tested applications. This fact reinforces our approach for the in-depth analysis, which scans the HTTP request traffic. With our analysis, we uncovered that 58% of the top 50 free applications expose information. Although we did not the detected the exposure of highly sensitive information like phone numbers or passwords, we revealed that applications commonly share unique device identifiers (45.4%) and the location (12.6%) with advertising companies. Especially the alarming fact that some of the most popular Android applications like the Cut the Rope Free application, shared the most information with advertising companies confirms the importance of the network traffic analysis.

For the known malicious applications we were only able to gain general knowledge about the communication with the automated traffic analysis. Accordingly, to directly identify malicious behaviour, the thorough manual analysis by an expert would be required. Still, we found out that even malware frequently (70%) integrates advertising. Beside the exposure of unique identifiers (51.2%) and the location 14.7% , 11% of the malicious application even exposed the phone number to advertisers.

We want to emphasise that many applications *do not provide any notice to the user* for their heavy advertising habits, and even if it is denoted in the EULA, it is hard for the user to understand the consequences. The transmission of unique device identifiers allows advertisers to *track the behaviour of the users* over time. Together with information like the location or even the phone number it allows to reveal quite personal facts about the user. For example, it could unveil where you live, where you work, who is your employer and what you commonly do in your free time. Although the information might not be directly related to you as a individual person, it already shows the implications of such advertising behaviour and the *danger for the privacy of the user*. We even more want to stress, if advertises at one point get to more detailed user information, they can relate it to your whole history using the unique device identifiers.

Although we *could not directly identify malicious applications* with our quite simple network traffic analysis approach, we were already able to reveal interesting facts about the advertising habits of Android applications. Still, we want to mention that our analysis method has *certain limitations*. We currently only analyse the HTTP request parameters and header fields for the exposure of information. We apply direct search for the limited set of information provided with test environment properties. Furthermore, we only imitate a limited number of obfuscation methods. Accordingly, the applications might have exposed even more information, which we just did not detected with our analysis method. Moreover,

in our case study we did not apply any man-in-the-middle attacks, to analyse the secure communication of the applications. In the next section, we will mention some possible improvements and give some thoughts on the future work.

8.2 Future Work

To improve the data exposure detection capabilities of NF4Droid, it would be necessary to imitate more obfuscation methods. Additionally, the investigation of more test environment properties could reveal exposure of further information. Regarding to the collecting of the test environment properties, which currently has to be done manually by the user, we propose the development of a small Android application. The application could automatically extract all the necessary informations and pass them to NF4Droid.

Since the manual testing and capturing of the applications is a time-consuming process, we reference to other projects, which automate the testing of applications. Automated testing methods could easily capture the network traffic and the test environment properties during the test run of an application, accordingly, a large number of applications could subsequently be analysed with NF4Droid.

The use of external capturing methods and the deployment of man-in-the-middle attacks could reveal the exposure of sensitive information from applications using secure communication. However, certain limitations exist for this approach, depending on the implementation and security measures of the applications.

NF4Droid is currently designed for the analysis of the network traffic from Android applications. However, it could easily be extended for the use with other mobile platforms like iOS or Windows Phone. The according possibilities for the capturing of the network would need to be evaluated and the test environment properties, used during the in-depth analysis, would need to be adapted.

NF4Droid provides a first implementation of a network traffic analysis method for Android applications. It demonstrates the general potential of the network traffic analysis and builds a base for the development of further analysis methods.

Appendix A

Acronyms

OS Operating System

GPS Global Positioning System

MSISDN Mobile Subscriber Integrated Services Digital Network Number

CC Country Code

NDC National Destination Code

SN Subscriber Number

SIM Subscriber Identity Module

MCC Mobile Country Code

MNC Mobile Network Code

LAC Location Area Code

GSM Global System for Mobile Communications

LA Location Area

LAI Location Area Identity

CID Cell-ID

ISO International Organization for Standardization

SPN Service Provider Name

IMSI International Mobile Subscriber Identity

MSIN Mobile Subscriber Identification Number

IMEI International Mobile Equipment Identity

EIR Equipment Identity Register

ICCID Integrated Circuit Card Identifier

SSID Service Set Identification

BSSID Basic Service set identification

WLAN Wireless Local Area Network

MAC Media Access Control

AP Access Point

IP Internet Protocol

A-GPS Assisted GPS

SIP Session Initiation Protocol

URL Uniform Resource Locator

SMS Short Message

MMS Multimedia Message

SD Secure Digital

GLS Google Location Server

UID Unique User Identifier

CDMA Code Division Multiple Access

API Application Programming Interface

NF4Droid Network Forensics for Android

RIM Research In Motion

SDK Software Development Kit

EULA end-user license agreement

C&C command and control

DDoS distributed denial-of-service

APK Android application package

GWT Google Web Toolkit

OTA Over-the-air

IPC inter-process communication

XML Extensible Markup Language

DEX Dalvik Executable

CFG control flow graph

VPN Virtual Private Network

SMTP Simple Mail Transfer Protocol

PCAP packet capture

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

TCP Transmission Control Protocol

UDP User Datagram Protocol

adb Android Debug Bridge

GWT Google Web Toolkit

AJAX Asynchronous JavaScript and XML

RCP remote procedure call

UI User Interface

SQL Structured Query Language

JPA Java Persistence API

ORM object-relational mapping

DDL data definition language

RDBMS relational database management system

SVG Scalable Vector Graphics

VML Vector Markup Language

HTML HyperText Markup Language

RPC Remote Procedure Call

RMI Remote Method Invocation

DTO Data Transfer Object

MVP Model View Presenter

OSI Open Systems Interconnection

SQL Structured Query Language

AOP Aspect-Oriented Programming

POM Project Object Model

DB database

Bibliography

[1] Adel Youssef, Arunesh Mishra [2008]. *My Location now with Wi-Fi - Official Google Mobile Blog*. http://googlemobile.blogspot.com/2008/10/my-location-now-with-wi-fi.html. Last visited on 2011-10-14. (Cited on page 44.)

[2] AdMob [2012]. *AdWhirl*. https://www.adwhirl.com/. Last visited on 2011-10-14. (Cited on pages 9 and 50.)

[3] Amazon [2012]. *Appstore for Android*. http://www.amazon.com/appstore. Last visited on 2011-10-14. (Cited on page 14.)

[4] Android [2012]. *Account | Android Developers*. https://developer.android.com/reference/android/accounts/Account.html. Last visited on 2011-10-14. (Cited on page 45.)

[5] Android [2012]. *Android Security Overview | Android Open Source*. http://source.android.com/tech/security/index.html. Last visited on 2011-10-14. (Cited on pages 7 and 12.)

[6] Android [2012]. *The AndroidManifest.xml File | Android Developers*. https://developer.android.com/guide/topics/manifest/manifest-intro.html. Last visited on 2011-10-14. (Cited on page 22.)

[7] Android [2012]. *AppInfo | Android Developers*. https://developer.android.com/reference/android/content/pm/ApplicationInfo.html. Last visited on 2011-10-14. (Cited on page 49.)

[8] Android [2012]. *Browser.BookmarkColumns | Android Developers*. https://developer.android.com/reference/android/provider/Browser.BookmarkColumns.html. Last visited on 2011-10-14. (Cited on page 47.)

[9] Android [2012]. *Build | Android Developers*. https://developer.android.com/reference/android/os/Build.html. Last visited on 2011-10-14. (Cited on page 42.)

[10] Android [2012]. *Build.VERSION | Android Developers*. https://developer.android.com/reference/android/os/Build.VERSION.html. Last visited on 2011-10-14. (Cited on page 42.)

[11] Android [2012]. *CalendarContract | Android Developers*. https://developer.android.com/reference/android/provider/CalendarContract.html. Last visited on 2011-10-14. (Cited on page 46.)

[12] Android [2012]. *CallLog.Calls | Android Developers*. https://developer.android.com/reference/android/provider/CallLog.Calls.html. Last visited on 2011-10-14. (Cited on page 48.)

[13] Android [2012]. *ContactsContract | Android Developers*. https://developer.android.com/reference/android/provider/ContactsContract.html. Last visited on 2011-10-14. (Cited on page 46.)

[14] Android [2012]. *ContentProvider | Android Developers*. https://developer.android.com/reference/android/content/ContentProvider.html. Last visited on 2011-10-14. (Cited on page 49.)

[15] Android [2012]. *Data Storage | Android Developers*. https://developer.android.com/guide/topics/data/data-storage.html. Last visited on 2011-10-14. (Cited on page 48.)

[16] Android [2012]. *Designing for Security | Android Developers*. https://developer.android.com/guide/practices/security.html#UserData. Last visited on 2011-10-14. (Cited on page 45.)

[17] Android [2012]. *Locale | Android Developers*. https://developer.android.com/reference/java/util/Locale.html. Last visited on 2011-10-14. (Cited on page 42.)

[18] Android [2012]. *Location | Android Developers*. https://developer.android.com/reference/android/location/Location.html. Last visited on 2011-10-14. (Cited on page 44.)

[19] Android [2012]. *NetworkInterface | Android Developers*. https://developer.android.com/reference/java/net/NetworkInterface.html. Last visited on 2011-10-14. (Cited on page 44.)

[20] Android [2012]. *Obtaining User Location | Android Developers*. https://developer.android.com/guide/topics/location/obtaining-user-location.html. Last visited on 2011-10-14. (Cited on page 44.)

[21] Android [2012]. *Package Info | Android Developers*. https://developer.android.com/reference/android/content/pm/PackageInfo.html. Last visited on 2011-10-14. (Cited on page 49.)

[22] Android [2012]. *Settings.Secure | Android Developers*. https://developer.android.com/reference/android/provider/Settings.Secure.html#ANDROID_ID. Last visited on 2011-10-14. (Cited on page 41.)

[23] Android [2012]. *Signing Your Applications | Android Developers*. https://developer.android.com/guide/publishing/app-signing.html. Last visited on 2011-10-14. (Cited on pages 13 and 49.)

[24] Android [2012]. *SmsMessage | Android Developers*. https://developer.android.com/reference/android/telephony/SmsMessage.html. Last visited on 2011-10-14. (Cited on page 48.)

[25] Android [2012]. *System | Android Developers*. https://developer.android.com/reference/java/lang/System.html#getProperty(java.lang.String). Last visited on 2011-10-14. (Cited on page 42.)

[26] Android [2012]. *TelephonyManager | Android Developers*. https://developer.android.com/reference/android/telephony/TelephonyManager.html. Last visited on 2011-10-14. (Cited on page 42.)

[27] Android [2012]. *WifiInfo | Android Developers*. https://developer.android.com/reference/android/net/wifi/WifiInfo.html. Last visited on 2011-10-14. (Cited on page 43.)

[28] Android Developers [2012]. *Manifest.permission | Android Developers*. https://developer.android.com/reference/android/Manifest.permission.html. Last visited on 2011-10-14. (Cited on page 41.)

[29] Android Developers [2012]. *ProGuard | Android Developers*. https://developer.android.com/tools/help/proguard.html. Last visited on 2011-10-14. (Cited on page 23.)

[30] Android Developers [2012]. *Reading and Writing Logs| Android Developers*. https://developer.android.com/guide/developing/debugging/debugging-log.html. Last visited on 2011-10-14. (Cited on page 47.)

[31] Android Developers [2012]. *Security and Permissions | Android Developers*. https://developer.android.com/guide/topics/security/security.html. Last visited on 2011-10-14. (Cited on page 41.)

[32] Anthony Desnos [2012]. *androguard - Reverse engineering, Malware and goodware analysis of Android applications*. https://code.google.com/p/androguard/. Last visited on 2011-10-14. (Cited on pages 22, 23 and 24.)

[33] AppBrain [2012]. *AppBrain*. http://www.appbrain.com/. Last visited on 2011-10-14. (Cited on page 14.)

[34] Apple [2012]. *Apple - iOS 5*. https://www.apple.com/ios/. Last visited on 2011-10-14. (Cited on page 1.)

[35] Apple [2012]. *See apps and games from the App Store*. https://www.apple.com/uk/iphone/from-the-app-store/. Last visited on 2011-10-14. (Cited on pages 1 and 13.)

[36] Apple Developer [2012]. *App Store Review Guidelines*. https://developer.apple.com/appstore/guidelines.html. Last visited on 2011-10-14. (Cited on page 8.)

[37] Apple Inc. [2012]. *Apple Answers the FCC's Questions*. https://www.apple.com/hotnews/apple-answers-fcc-questions/. Last visited on 2011-10-14. (Cited on pages 2 and 14.)

[38] Apple Inc. [2012]. *iOS Security*. http://images.apple.com/ipad/business/docs/iOS_Security_May12.pdf. Last visited on 2011-10-14. (Cited on pages 7 and 12.)

[39] Apple Inc. [2012]. *Technical Note TN2250*. https://developer.apple.com/library/ios/#technotes/tn2250/_index.html. Last visited on 2011-10-14. (Cited on page 13.)

[40] Apvrille, Axelle [2011]. *Cryptography for mobile malware obfuscation*. In *RSA Conference Europe*. October. http://365.rsaconference.com/docs/DOC-3039. (Cited on page 23.)

[41] AS, Highsoft Solutions [2012]. *Highcharts - Interactive JavaScript charts for your webpage*. http://www.highcharts.com/. Last visited on 2011-10-14. (Cited on page 75.)

[42] Au, Kathy Wain Yee, Yi Fan Zhou, Zhen Huang, Phillipa Gill, and David Lie [2011]. *Short Paper: A Look at SmartPhone Permission Models*. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices - SPSM '11*, page 63. SPSM '11, ACM Press, New York, New York, USA. ISBN 9781450310000. doi:10.1145/2046614.2046626. http://dl.acm.org/citation.cfm?doid=2046614.2046626. (Cited on page 13.)

[43] Basic4Android [2012]. *Basic4android - GPS*. http://www.basic4ppc.com/android/help/gps.html. Last visited on 2011-10-14. (Cited on page 44.)

[44] Ben Gruver [2012]. *smali - An assembler/disassembler for Android's dex format*. https://code.google.com/p/smali/. Last visited on 2011-10-14. (Cited on page 22.)

[45] Benco, David S., Paresh C. Kanabar, John C. V. Nguyen, and Huixian Song [2008]. *US Patent Application Publicaiton: Caller ID Spoofing*. (Cited on page 43.)

[46] Beresford, Alastair R, Andrew Rice, and Nicholas Skehin [2011]. *MockDroid : trading privacy for application functionality on smartphones Categories and Subject Descriptors*. *Manager*. http://www.cl.cam.ac.uk/~acr31/pubs/beresford-mockdroid.pdf. (Cited on pages 21 and 25.)

[47] Blaesing, Thomas, Leonid Batyuk, Aubrey-derrick Schmidt, Seyit Ahmet Camtepe, and Sahin Albayrak [2010]. *An Android Application Sandbox system for suspicious software detection*. In *2010 5th International Conference on Malicious and Unwanted Software*, pages 55–62. IEEE, IEEE, Berlin. ISBN 978-1-4244-9353-1. doi:10.1109/MALWARE.2010.5665792. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5665792. (Cited on pages 24 and 26.)

[48] Brian X. Chen, Nick Bilton [2012]. *Et Tu, Google? Android Apps Can Also Secretly Copy Photos - NYTimes.com*. http://bits.blogs.nytimes.com/2012/03/01/android-photos/. Last visited on 2011-10-14. (Cited on page 49.)

[49] Brixx, Todd [2012]. *Four ways we're improving Marketplace*. http://windowsteamblog.com/windows_phone/b/wpdev/archive/2012/04/30/four-ways-we-re-improving-marketplace.aspx. Last visited on 2011-10-14. (Cited on page 14.)

[50] Carlos Castillo [2011]. *Spitmo vs Zitmo: Banking Trojans Target Android - McAfee Labs*. http://blogs.mcafee.com/mcafee-labs/spitmo-vs-zitmo-banking-trojans-target-android. Last visited on 2011-10-14. (Cited on page 10.)

[51] Charles, Ryan [2010]. *Android Market Ranking Algorithm: The New Black Box*. http://ryenyc.tumblr.com/post/942264066/android-market-ranking-algorithm-black-box. Last visited on 2011-10-14. (Cited on page 77.)

[52] Charles Arthur [2012]. *Android over 50% of smartphone sales as Nokia and RIM feel strain | Technology | guardian.co.uk*. http://www.guardian.co.uk/technology/2012/may/16/android-smartphone-market-50-percent. Last visited on 2011-10-14. (Cited on page 1.)

[53] Chin, Erika, Adrienne Porter Felt, Kate Greenwood, and David Wagner [2011]. *Analyzing Inter-Application Communication in Android*. In *MobiSys*, pages 239–252. MobiSys '11, ACM Press. ISBN 9781450306430. doi:10.1145/1999995.2000018. http://www.eecs.berkeley.edu/~emc/papers/mobi168-chin.pdf. (Cited on pages 20 and 24.)

[54] Co., Taosoftware [2011]. *tPacketCapture*. http://www.taosoftware.co.jp/en/android/packetcapture/. Last visited on 2011-10-14. (Cited on pages 28 and 37.)

[55] Commons, Creative [2012]. *Attribution-NonCommercial 3.0 Unported - CC BY-NC 3.0*. https://creativecommons.org/licenses/by-nc/3.0/. Last visited on 2011-10-14. (Cited on page 75.)

[56] Commons, Creative [2012]. *Attribution-ShareAlike 3.0 Unported - CC BY-SA 3.0*. https://creativecommons.org/licenses/by-sa/3.0/. Last visited on 2011-10-14. (Cited on page 75.)

[57] Cong Zheng [2012]. *apkinspector - APKinspector is a powerful GUI tool for analysts to analyze the Android applications*. https://code.google.com/p/apkinspector/. Last visited on 2011-10-14. (Cited on page 23.)

[58] contagio [2012]. *contagio mobile*. http://contagiominidump.blogspot.co.at/. Last visited on 2011-10-14. (Cited on page 84.)

[59] Corporation, Symantec [2012]. *Norton Mobile Security*. http://us.norton.com/norton-mobile-security/. Last visited on 2011-10-14. (Cited on pages 8 and 21.)

[60] Corporation, ZTE [2012]. *ZTE Corporation*. http://wwwen.zte.com.cn/en/. Last visited on 2011-10-14. (Cited on pages 77 and 78.)

[61] Cortesi [2011]. *How UDIDs are used: a survey*. http://corte.si/posts/security/apple-udid-survey/. Last visited on 2011-10-14. (Cited on page 29.)

[62] Cortesi [2011]. *mitmproxy*. http://mitmproxy.org/. Last visited on 2011-10-14. (Cited on page 29.)

[63] CyanogenMod [2012]. *CyanogenMod | Android Community Rom*. http://www.cyanogenmod.com/. Last visited on 2011-10-14. (Cited on page 77.)

[64] Dan Goodin [2009]. *Backdoor in top iPhone games stole user data, suit claims*. http://www.theregister.co.uk/2009/11/06/iphone_games_storm8_lawsuit/. Last visited on 2011-10-14. (Cited on page 9.)

[65] Daniel Baeumges [2012]. *TaintDroid Runner - TaintDroid 2.3*. https://sites.google.com/site/taintdroid23/taintdroid_runner. Last visited on 2011-10-14. (Cited on page 26.)

[66] David Rogers [2011]. *blog.mobilephonesecurity.org: Voicemail hacking and the ṕhone hackingścandal - how it worked, questions to be asked and improvements to be made*. http://blog.mobilephonesecurity.org/2011/07/voicemail-hacking-and-phone-hacking.html. Last visited on 2011-10-14. (Cited on page 43.)

[67] DEF CON Communications [2012]. *DEF CON 19 Hacking Conference*. http://defcon.org/html/defcon-19/dc-19-index.html. Last visited on 2011-10-14. (Cited on page 3.)

[68] Denis Maslennikov [2012]. *Find and Call: Leak and Spam - Securelist*. https://www.securelist.com/en/blog/208193641/Find_and_Call_Leak_and_Spam. Last visited on 2011-10-14. (Cited on page 14.)

[69] Developers, Android [2012]. *Android Debug Bridge | Android Developers*. https://developer.android.com/tools/help/adb.html. Last visited on 2011-10-14. (Cited on page 77.)

[70] Developers, Android [2012]. *VpnService*. https://developer.android.com/reference/android/net/VpnService.html. Last visited on 2011-10-14. (Cited on pages 36 and 37.)

[71] Developers, Google [2012]. *Google Chart Tools*. https://developers.google.com/chart/. Last visited on 2011-10-14. (Cited on page 75.)

[72] Developers, Google [2012]. *Google Web Toolkit*. https://developers.google.com/web-toolkit/. Last visited on 2011-10-14. (Cited on page 73.)

[73] Dhar, Subhankar and Upkar Varshney [2011]. *Challenges and business models for mobile location-based services and advertising*. *Communications of the ACM*, 54(5), page 121. ISSN 00010782. doi:10.1145/1941487.1941515. http://dl.acm.org/citation.cfm?id=1941515. (Cited on page 45.)

[74] eLinux.org [2012]. *Android Logging System - eLinux.org*. http://elinux.org/Android_Logging_System. Last visited on 2011-10-14. (Cited on page 47.)

[75] Emmanuel Dupuy [2012]. *JD | Java Decompiler*. http://java.decompiler.free.fr/. Last visited on 2011-10-14. (Cited on page 23.)

[76] Enck, William, Landon P Cox, Peter Gilbert, and Patrick Mcdaniel [2010]. *TaintDroid : An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones*. In *9th USENIX conference on Operating systems design and implementation*. Berkley. http://www.usenix.org/event/osdi10/tech/full_papers/Enck.pdf. (Cited on page 25.)

[77] Enck, William, Damien Octeau, Patrick Mcdaniel, and Swarat Chaudhuri [2011]. *A Study of Android Application Security*. In *20th USENIX Security Symposium*. San Francisco. http://www.enck.org/pubs/enck-sec11.pdf. (Cited on pages 22 and 23.)

[78] Enck, William, Machigar Ongtang, and Patrick McDaniel [2009]. *On lightweight mobile phone application certification*. *Security*, pages 235–245. ISSN 15437221. doi:10.1145/1653662.1653691. http://portal.acm.org/citation.cfm?doid=1653662.1653691. (Cited on page 23.)

[79] F-Secure [2012]. *Mobile Threat Report*. Technical Report, F-Secure. http://www.f-secure.com/weblog/archives/MobileThreatReport_Q1_2012.pdf. (Cited on page 8.)

[80] Felt, Adrienne Porter, Matthew Finifter, Erika Chin, Steven Hanna, and David Wagner [2011]. *A Survey of Mobile Malware in the Wild*. Technical Report, University of California, Berkley. http://dl.acm.org/citation.cfm?id=2046618. (Cited on pages 8, 10 and 11.)

[81] Felt, Adrienne Porter, Greenwood Kate, and David Wagner [2010]. *The Effectiveness of Install-Time Permission Systems for Third-Party Applications*. Technical Report, University of California, Berkeley. http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-143.pdf. (Cited on pages 7 and 41.)

[82] Felt, A.P., Erika Chin, Steve Hanna, Dawn Song, and David Wagner [2011]. *Android permissions demystified*. Technical Report, Technical Report UCB/EECS-2011-48, University of California, Berkeley. http://www.eecs.berkeley.edu/~emc/papers/EECS-2011-48.pdf. (Cited on page 23.)

[83] forensic blog [2012]. *forensic blog - Current Android Malware*. http://forensics.spreitzenbarth.de/android-malware/. Last visited on 2011-10-14. (Cited on page 84.)

[84] Fossi, Marc, Eric Johnson, and David Mckinney [2008]. *Symantec Report on the Underground Economy*. Technical Report november, Symantec. http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_underground_economy_report_11-2008-14525717.en-us.pdf. (Cited on page 10.)

[85] Foundation, Free Software [2012]. *GNU Library General Public License v2.0*. https://www.gnu.org/licenses/old-licenses/lgpl-2.0.html. Last visited on 2011-10-14. (Cited on page 75.)

[86] Foundation, Free Software [2012]. *GNU Library General Public License v2.1*. https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html. Last visited on 2011-10-14. (Cited on page 74.)

[87] Foundation, The Apache Software [2012]. *Apache License, Version 2.0*. https://www.apache.org/licenses/LICENSE-2.0.html. Last visited on 2011-10-14. (Cited on pages 73, 74, 75 and 76.)

[88] Foundation, The Apache Software [2012]. *Maven*. https://maven.apache.org/. Last visited on 2011-10-14. (Cited on page 76.)

[89] Foundation, The Eclipse [2012]. *The AspectJ Project*. http://eclipse.org/aspectj/. Last visited on 2011-10-14. (Cited on page 75.)

[90] Foundation, The Eclipse [2012]. *Eclipse Public License - v 1.0*. http://eclipse.org/legal/epl-v10.html. Last visited on 2011-10-14. (Cited on page 75.)

[91] Fuchs, Adam P, Avik Chaudhuri, and Jeffrey S Foster [2010]. *SCanDroid : Automated Security Certification of Android Applications*. *Read*, 10, page 328. doi:10.1.1.164.6899. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.148.2511&rep=rep1&type=pdf. (Cited on page 24.)

[92] Fulton, Eric [2011]. *Cellular Privacy - A Forensic Analysis of Android Network Traffic*. In *DEF CON 19*. Las Vegas. http://www.triskt.com/research/Eric_Fulton_Defcon_2011_export.pdf. (Cited on pages 3, 29, 30, 33, 78 and 79.)

[93] G Data Software AG [2012]. *Malware Or Not Malware - That's The Question*. https://blog.gdatasoftware.com/blog/article/malware-or-not-malware-thats-the-question.html. Last visited on 2011-10-14. (Cited on page 10.)

[94] Gartner Inc. [2011]. *Gartner Says Android to Command Nearly Half of Worldwide Smartphone Operating System Market by Year-End 2012*. http://www.gartner.com/it/page.jsp?id=1622614. Last visited on 2011-10-14. (Cited on page 1.)

[95] Gartner Inc. [2012]. *Gartner: Sales of Mobile Phones 2012*. http://www.gartner.com/it/page.jsp?id=2017015. Last visited on 2011-10-14. (Cited on pages 1, 7 and 15.)

[96] Gartner Inc. [2012]. *Gartner Says Worldwide Media Tablets Sales to Reach 119 Million Units in 2012*. http://www.gartner.com/it/page.jsp?id=1980115. Last visited on 2011-10-14. (Cited on page 1.)

[97] GetJar [2012]. *GetJar*. http://www.getjar.com/. Last visited on 2011-10-14. (Cited on page 14.)

[98] Google [2012]. *Android*. http://www.android.com/. Last visited on 2011-10-14. (Cited on pages vi, 1, 79 and 81.)

[99] Google [2012]. *Android Apps on Google Play*. https://play.google.com/. Last visited on 2011-10-14. (Cited on pages 1 and 13.)

[100] Google [2012]. *Chrome Browser*. https://www.google.com/intl/en/chrome/browser/. Last visited on 2011-10-14. (Cited on page 64.)

[101] Google [2012]. *Location-based services*. https://support.google.com/maps/bin/answer.py?hl=en&answer=1725632. Last visited on 2011-10-14. (Cited on page 44.)

[102] GoogleDevelopers [2012]. *Declarative Layout with UiBinder - GWT*. https://developers.google.com/web-toolkit/doc/2.4/DevGuideUiBinder. Last visited on 2011-10-14. (Cited on page 65.)

[103] GoogleDevelopers [2012]. *Development with Activities and Places - GWT*. https://developers.google.com/web-toolkit/doc/latest/DevGuideMvpActivitiesAndPlaces. Last visited on 2011-10-14. (Cited on page 65.)

[104] GoogleDevelopers [2012]. *Getting Started with RequestFactory - GWT*. https://developers.google.com/web-toolkit/doc/2.4/DevGuideRequestFactory. Last visited on 2011-10-14. (Cited on page 65.)

[105] GoogleDevelopers [2012]. *Google APIs Terms of Service*. https://developers.google.com/terms/. Last visited on 2011-10-14. (Cited on page 75.)

[106] GoogleDevelopers [2012]. *Making Remote Procedure Calls - GWT*. https://developers.google.com/web-toolkit/doc/latest/tutorial/RPC. Last visited on 2011-10-14. (Cited on page 64.)

[107] GoogleDevelopers [2012]. *Using the Google Chart Tools with GWT*. https://code.google.com/p/gwt-google-apis/wiki/VisualizationGettingStarted. Last visited on 2011-10-14. (Cited on page 75.)

[108] Grace, Michael, Wu Zhou, Xuxian Jiang, and Ahmad-reza Sadeghi [2011]. *Unsafe Exposure Analysis of Mobile In-App Advertisements*. *Security*, 067(Section 2). http://www.csc.ncsu.edu/faculty/jiang/pubs/WISEC12_ADRISK.pdf. (Cited on page 45.)

[109] Grace, Michael, Yajin Zhou, Zhi Wang, Xuxian Jiang, and Oval Drive [2012]. *Systematic Detection of Capability Leaks in Stock Android Smartphones*. *North*. http://www.csc.ncsu.edu/faculty/jiang/pubs/NDSS12_WOODPECKER.pdf. (Cited on pages 20 and 24.)

[110] Group, Moxie [2012]. *GWT Highcharts*. http://www.moxiegroup.com/moxieapps/gwt-highcharts/. Last visited on 2011-10-14. (Cited on page 75.)

[111] GWT-Bootstrap [2012]. *GWT-Bootstrap*. http://gwtbootstrap.github.com/. Last visited on 2011-10-14. (Cited on page 75.)

[112] Harry, Sverdlove [2011]. *The Most Vulnerable Smartphones of 2011*. Technical Report, Bit9. https://www.bit9.com/files/Bit9Report_SmartPhones2011.pdf. (Cited on page 42.)

[113] Hat, Red [2012]. *Hibernate - JBoss Community*. http://www.hibernate.org/. Last visited on 2011-10-14. (Cited on page 74.)

[114] Hiroshi Lockheimer [2012]. *Android and Security*. http://googlemobile.blogspot.co.at/2012/02/android-and-security.html. Last visited on 2011-10-14. (Cited on pages 2, 14 and 26.)

[115] Hornyack, Peter, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall [2011]. *These aren't the droids you're looking for*. In *Proceedings of the 18th ACM conference on Computer and communications security - CCS '11*, page 639. CCS '11, Microsoft Research, ACM Press, New York, New York, USA. ISBN 9781450309486. doi:10.1145/2046707.2046780. http://dl.acm.org/citation.cfm?doid=2046707.2046780. (Cited on pages 21 and 25.)

[116] Idika, Nwokedi [2007]. *A Survey of Malware Detection Techniques*. *Purdue University*, page 48. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.75.4594&rep=rep1&type=pdf. (Cited on page 22.)

[117] Inc., MaxMind [2012]. *GeoLite Databases*. https://www.maxmind.com/en/geolite. Last visited on 2011-10-14. (Cited on page 74.)

[118] Intrepidus Group [2012]. *Mallory: Transparent TCP and UDP Proxy*. http://intrepidusgroup.com/insight/mallory/. Last visited on 2011-10-14. (Cited on page 29.)

[119] iPhone Dev Team [2012]. *Dev-Team Blog*. http://blog.iphone-dev.org/. Last visited on 2011-10-14. (Cited on page 14.)

[120] ISO [2012]. *Publicly Available Standards - ISO/IEC 7498-1:1994*. http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html. Last visited on 2011-10-14. (Cited on page 66.)

[121] ISO - International Organization for Standardization [2012]. *ISO - Maintenance Agency for ISO 3166 country codes - ISO 3166-2*. http://www.iso.org/iso/country_codes/background_on_iso_3166/iso_3166-2.htm. Last visited on 2011-10-14. (Cited on page 43.)

[122] ITU - International Telecommunication Union [2012]. *E.164: The international public telecommunication numbering plan*. https://www.itu.int/rec/T-REC-E.164/. Last visited on 2011-10-14. (Cited on page 42.)

[123] ITU - International Telecommunication Union [2012]. *E.212: The international identification plan for public networks and subscriptions*. https://www.itu.int/rec/T-REC-E.212/. Last visited on 2011-10-14. (Cited on page 42.)

[124] Ji Zhengrong, Jain Ravi [2008]. *Google enables Location-aware Applications for 3rd Party Developers - Official Google Mobile Blog*. http://googlemobile.blogspot.com/2008/06/google-enables-location-aware.html. Last visited on 2011-10-14. (Cited on page 44.)

[125] Jon Fingas [2012]. *Google Play hits 600,000 apps, 20 billion total installs - Engadget*. http://www.engadget.com/2012/06/27/google-play-hits-600000-apps/. Last visited on 2011-10-14. (Cited on pages 1 and 15.)

[126] Jordan Golson [2012]. *Apple Requires User Permission Before Apps Can Access Personal Data in iOS 6 - Mac Rumors*. http://www.macrumors.com/2012/06/14/apple-requires-user-permission-before-apps-can-access-personal-data-in-ios-6/. Last visited on 2011-10-14. (Cited on page 13.)

[127] Jumptap [2012]. *Jumptap - The leader in targeted mobile advertising*. http://www.jumptap.com/. Last visited on 2011-10-14. (Cited on page 9.)

[128] Juniper Networks [2012]. *2011 Mobile Threats Report*. Technical Report February, Juniper Networks. https://www.juniper.net/us/en/local/pdf/additional-resources/jnpr-2011-mobile-threats-report.pdf. (Cited on pages 1, 2, 7, 15 and 16.)

[129] Kraken [2012]. *Kraken*. http://krakenapps.org/. Last visited on 2011-10-14. (Cited on page 74.)

[130] Lab, Kaspersky [2012]. *Kaspersky Mobile Security*. http://www.kaspersky.com/kaspersky_mobile_security. Last visited on 2011-10-14. (Cited on page 21.)

[131] Lemoine, Hanno [2012]. *The DyAnA Framework*. In *SIGINT*. (Cited on page 26.)

[132] Lineberry, Anthony, David Luke Richardson, and Tim Wyatt [2010]. *THESE AREN'T THE PERMISSIONS YOU'RE LOOKING FOR*. In *DEF CON 18*. Las Vegas. https://www.defcon.org/images/defcon-18/dc-18-presentations/Lineberry/DEFCON-18-Lineberry-Not-The-Permissions-You-Are-Looking-For.pdf. (Cited on page 48.)

[133] Lookout, Inc. [2012]. *Droid Mobile Security & Security for all Smartphones*. https://www.mylookout.com/. Last visited on 2011-10-14. (Cited on pages 8 and 21.)

[134] Matenaar, Felix, Patrick Schulz, Andreas Galauner, and Mark Schlösser [2012]. *Android Analysis Framework dexter*. In *SIGINT*. https://program.sigint.ccc.de/fahrplan/system/attachments/27/original/dexlabs.pdf. (Cited on page 24.)

[135] MaxMind [2012]. *MaxMind APIs*. https://dev.maxmind.com/geoip/downloadable. Last visited on 2011-10-14. (Cited on page 75.)

[136] McAfee [2012]. *McAfee Threats Report : First Quarter 2012*. Technical Report, McAfee Labs. http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q1-2012.pdf. (Cited on page 7.)

[137] McDaniel, Patrick and William Enck [2010]. *Not So Great Expectations: Why Application Markets Haven't Failed Security*. *IEEE Security & Privacy Magazine*, pages 76–78. http://www.patrickmcdaniel.org/pubs/sp-markets10.pdf. (Cited on pages 13 and 14.)

[138] Meissner, Wayne [2012]. *jnr-netdb*. https://github.com/wmeissner/jnr-netdb. Last visited on 2011-10-14. (Cited on page 74.)

[139] Microsoft [2012]. *Application Certification Requirements for Windows Phone*. http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh184843(v=vs.92). Last visited on 2011-10-14. (Cited on pages 13 and 14.)

[140] Microsoft [2012]. *Marketplace*. http://www.windowsphone.com/en-US/marketplace. Last visited on 2011-10-14. (Cited on pages 1, 7 and 13.)

[141] Microsoft [2012]. *Security for Windows Phone*. http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402533(v=vs.92). Last visited on 2011-10-14. (Cited on page 12.)

[142] Microsoft [2012]. *Windows Phone*. http://www.microsoft.com/windowsphone/. Last visited on 2011-10-14. (Cited on page 1.)

[143] Mitchell, Stewart [2010]. *Microsoft details Windows Phone 7 kill switch - PC Pro*. http://www.pcpro.co.uk/news/security/362485/microsoft-details-windows-phone-7-kill-switch. Last visited on 2011-10-14. (Cited on page 14.)

[144] MoÃ±ino, Manolo Carrasco [2012]. *GWTUpload*. https://code.google.com/p/gwtupload/. Last visited on 2011-10-14. (Cited on page 74.)

[145] MobiStealth [2012]. *Cell phone spy and monitoring software*. http://www.mobistealth.com/. Last visited on 2011-10-14. (Cited on pages 8 and 11.)

[146] Moxie Marlinspike [2012]. *sslstrip*. http://www.thoughtcrime.org/software/sslstrip/. Last visited on 2011-10-14. (Cited on page 29.)

[147] Nauman, Mohammad, Sohail Khan, and Xinwen Zhang [2010]. *Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints*. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security - ASIACCS '10*, page 328. Number c in ASIACCS '10, ACM Press, New York, USA. ISBN 9781605589367. doi:10.1145/1755688.1755732. http://portal.acm.org/citation.cfm?doid=1755688.1755732. (Cited on pages 21 and 25.)

[148] Nick Wingfield [2008]. *IPhone Software Sales Take Off: Apple's Jobs*. http://online.wsj.com/article/SB121842341491928977.html. Last visited on 2011-10-14. (Cited on pages 2 and 14.)

[149] Oberheide, Jon and Charlie Miller [2012]. *Dissecting the Android Bouncer*. In *SummerCon*. New York, New York, USA. http://jon.oberheide.org/files/summercon12-bouncer.pdf. (Cited on pages 18 and 26.)

[150] O'Kane, Philip, Sakir Sezer, and Kieran McLaughlin [2011]. *Obfuscation: The Hidden Malware*. doi:10.1109/MSP.2011.98. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5975134. (Cited on pages 23 and 26.)

[151] Oracle [2012]. *JSR-000317 Java Persistence 2.0*. http://jcp.org/aboutJava/communityprocess/final/jsr317/index.html. Last visited on 2011-10-14. (Cited on page 74.)

[152] Oracle [2012]. *Trail: RMI*. http://docs.oracle.com/javase/tutorial/rmi/index.html. Last visited on 2011-10-14. (Cited on page 64.)

[153] Orthacker, Clemens, Peter Teufl, Stefan Kraxberger, Alexander Marsalek, Johannes Leibetseder, and Oliver Prevenhueber [2011]. *Android Security Permissions - Can we trust them. Accepted but not yet published at MobiSEC*. https://online.tugraz.at/tug_online/voe_main2.getvolltext?pCurrPk=57576. (Cited on page 20.)

[154] Panda Security [2009]. *Virus Encyclopedia - Eeki.A*. http://www.pandasecurity.com/homeusers/security-info/215107/Eeki.A. Last visited on 2011-10-14. (Cited on page 9.)

[155] Patrik Lantz [2011]. *droidbox - Android Application Sandbox*. https://code.google.com/p/droidbox/. Last visited on 2011-10-14. (Cited on page 26.)

[156] Paul Brodeur [2012]. *Zero-Permission Android Applications - Leviathan Security Group*. https://leviathansecurity.com/blog/archives/17-Zero-Permission-Android-Applications.html. Last visited on 2011-10-14. (Cited on page 41.)

[157] Paul Brodeur [2012]. *Zero-Permission Android Applications part 2 - Leviathan Security Group*. https://www.leviathansecurity.com/blog/archives/18-Zero-Permission-Android-Applications-part-2.html. Last visited on 2011-10-14. (Cited on page 41.)

[158] Pavel Kouznetsov [2012]. *JAD Java Decompiler*. http://www.varaneckas.com/jad/. Last visited on 2011-10-14. (Cited on page 23.)

[159] QueryDSL [2012]. *Querying SQL*. http://www.querydsl.com/static/querydsl/2.8.0/reference/html/ch02s03.html#d0e607. Last visited on 2011-10-14. (Cited on page 67.)

[160] QueryDSL, Mysema [2012]. *QueryDSL*. http://www.querydsl.com/. Last visited on 2011-10-14. (Cited on page 74.)

[161] Rafael Rivera, Long Zheng, Chris Walsh [2010]. *ChevronWP7*. http://www.chevronwp7.com/. Last visited on 2011-10-14. (Cited on page 14.)

[162] Ramu, Srikanth [2012]. *Mobile Malware Evolution, Detection and Defense*. http://blogs.ubc.ca/computersecurity/files/2012/04/SRamu_EECE572_SurveyPaper-SrikanthRamu.pdf. (Cited on page 11.)

[163] Research in Motion [2012]. *Cell Phones, Smartphones & Mobile Phones from BlackBerry.com*. http://worldwide.blackberry.com/. Last visited on 2011-10-14. (Cited on page 1.)

[164] Rich Cannings [2010]. *Exercising Our Remote Application Removal Feature*. http://android-developers.blogspot.co.at/2010/06/exercising-our-remote-application.html. Last visited on 2011-10-14. (Cited on pages 2 and 14.)

[165] Ryszard Wisniewski [2011]. *android-apktool - A tool for reverse engineering Android apk files*. https://code.google.com/p/android-apktool/. Last visited on 2011-10-14. (Cited on page 22.)

[166] Salameh, Khaled [2011]. *Windows Phone SMS attack discovered, reboots device and disables messaging hub - WinRumors*. http://www.winrumors.com/windows-phone-sms-attack-discovered-reboots-device-and-disables-messaging-hub/. Last visited on 2011-10-14. (Cited on page 16.)

[167] Schulz, Patrick [2012]. *Code Protection in Android*. https://net.cs.uni-bonn.de/fileadmin/user_upload/plohmann/2012-Schulz-Code_Protection_in_Android.pdf. (Cited on page 23.)

[168] Schwartzbach, Michael []. *Lecture Notes on Static Analysis*. http://www.itu.dk/people/brabrand/UFPE/Data-Flow-Analysis/static.pdf. (Cited on page 23.)

[169] Skiba Dmitry [2008]. *android4me - J2ME port of Google's Android*. https://code.google.com/p/android4me/. Last visited on 2011-10-14. (Cited on page 22.)

[170] Skyhook [2012]. *Skyhook: Location Technology*. http://www.skyhookwireless.com/location-technology/. Last visited on 2011-10-14. (Cited on page 44.)

[171] Slideme [2012]. *Slideme*. http://slideme.org/. Last visited on 2011-10-14. (Cited on page 14.)

[172] Smith, Eric [2010]. *iPhone Applications & Privacy Issues : An Analysis of Application Transmission of iPhone Unique Device Identifiers (UDIDs). New York*, pages 1–19. http://www.kompatscher.biz/phocadownload/iPhone-Applications-Privacy-Issues.pdf. (Cited on page 29.)

[173] Smule [2012]. *Magic Piano - Privacy Policy (updated March 27, 2012)*. https://www.smule.com/privacy. Last visited on 2011-10-14. (Cited on page 79.)

[174] SpringSource [2012]. *Spring Roo*. http://www.springsource.org/spring-roo. Last visited on 2011-10-14. (Cited on page 73.)

[175] SQLite [2012]. *SQLite Home Page*. https://www.sqlite.org/. Last visited on 2011-10-14. (Cited on page 49.)

[176] Symantec Corporation [2011]. *Android.Pjapps*. http://www.symantec.com/security_response/writeup.jsp?docid=2011-022303-3344-99. Last visited on 2011-10-14. (Cited on page 11.)

[177] Symantec Corporation [2011]. *Android.Smspacem*. http://www.symantec.com/security_response/writeup.jsp?docid=2011-052310-1322-99. Last visited on 2011-10-14. (Cited on page 9.)

[178] Symbian Foundation [2012]. *Symbian Foundation*. http://licensing.symbian.org/. Last visited on 2011-10-14. (Cited on page 1.)

[179] Tcpdump/Libpcap [2011]. *TCPDUMP/LIBPCAP public repository*. http://www.tcpdump.org/. Last visited on 2011-10-14. (Cited on pages 28, 36 and 37.)

[180] tele.ring [2012]. *tele.ring*. https://www.telering.at/. Last visited on 2011-10-14. (Cited on pages 77 and 78.)

[181] Teufl, Peter, Michaela Ferk, Andreas Fitzek, Daniel Hein, Stefan Kraxberger, and Clemens Orthacker [2012]. *Malware Detection by Applying Knowledge Discovery Processes to Application Metadata on the Android Market (Google Play)*. doi:10.1002/sec. (Cited on pages vii, 8, 15, 17, 18 and 27.)

[182] Teufl, Peter, Stefan Kraxberger, Clemens Orthacker, Alexander Marsalek, Johannes Leibetseder, and Oliver Prevenhueber [2011]. *Android Market Analysis with Activation Patterns*. *Work*. https://online.tugraz.at/tug_online/voe_main2.getvolltext?pCurrPk=57577. (Cited on page 13.)

[183] The Wall Street Journal [2010]. *What They Know - Mobile*. http://blogs.wsj.com/wtk-mobile/. Last visited on 2011-10-14. (Cited on pages 3, 28, 30, 33, 78 and 79.)

[184] Tim Bray [2011]. *Identifying App Installations*. http://android-developers.blogspot.com/2011/03/identifying-app-installations.html. Last visited on 2011-10-14. (Cited on page 42.)

[185] tutorialspoint [2012]. *GSM - Addresses and Identifiers (IMEI, IMSI, TMSI, LMSI, MSISDN, MSRN)*. http://www.tutorialspoint.com/gsm/gsm_addressing.htm. Last visited on 2011-10-14. (Cited on page 43.)

[186] Twitter [2012]. *Twitter Bootstrap*. http://twitter.github.com/bootstrap/. Last visited on 2011-10-14. (Cited on page 75.)

[187] UAS [2012]. *Spy Control*. https://play.google.com/store/apps/details?id=com.uas.smscontrol&hl=en. Last visited on 2011-10-14. (Cited on page 8.)

[188] Vennon, Troy [2010]. *Android Malware A Study of Known and Potential Malware Threats*. *Engineer*, pages 1–13. http://globalthreatcenter.com/wp-content/uploads/2010/03/Android-Malware-Whitepaper.pdf. (Cited on pages 8 and 14.)

[189] Vennon, Troy and David Stroop [2010]. *Android Market: Threat Analysis of the Android Market*. Technical Report, SMobile Systems, Columbus. http://threatcenter.smobilesystems.com/wp-content/uploads/2010/06/Android-Market-Threat-Analysis-6-22-10-v1.pdf. (Cited on pages 13 and 27.)

[190] Venugopal, Deepak and Guoning Hu [2008]. *Efficient signature based malware detection on mobile devices*. *Mobile Information Systems*, 4(1), pages 33–49. ISSN 1574017X. (Cited on page 22.)

[191] Wireshark Foundation [2012]. *Wireshark - Go deep.* http://www.wireshark.org/. Last visited on 2011-10-14. (Cited on pages 27, 29, 31 and 37.)

[192] Xiaobo Pan [2012]. *dex2jar - Tools to work with android .dex and java .class files*. https://code.google.com/p/dex2jar/. Last visited on 2011-10-14. (Cited on page 22.)

[193] Xuxian Jiang [2011]. *Questionable Android Apps - SndApps - Found and Removed from Official Android Market*. http://www.csc.ncsu.edu/faculty/jiang/SndApps/. Last visited on 2011-10-14. (Cited on page 10.)

[194] You, Ilsun and Kangbin Yim [2010]. *Malware Obfuscation Techniques: A Brief Survey*. In *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, pages 297–300. IEEE. ISBN 978-1-4244-8448-5. doi:10.1109/BWCCA.2010.85. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5633410. (Cited on page 23.)

[195] Zayed Rehman [2011]. *The Complete Review Of CyanogenMod 7*. http://www.addictivetips.com/mobile/the-complete-review-of-cyanogenmod-7-walkthrough-guide/#5. Last visited on 2011-10-14. (Cited on page 21.)

[196] ZeptoLab [2012]. *ZeptoLab - Privacy Policy*. http://www.zeptolab.com/pp.htm. Last visited on 2011-10-14. (Cited on page 80.)

[197] Zhou, Wu, Yajin Zhou, Xuxian Jiang, and Peng Ning [2012]. *Detecting repackaged smartphone applications in third-party android marketplaces*. In *Proceedings of the second ACM conference on Data and Application Security and Privacy - CODASKY '12*, page 317. 16th ACM Conference on Computer and Communications Security, CCS'09, ACM Press, New York, New York, USA. ISBN 9781450310918. ISSN 15437221. doi:10.1145/2133601.2133640. http://portal.acm.org/citation.cfm?doid=1653662.1653690. (Cited on page 24.)

[198] Zhou, Yajin [2012]. *Dissecting Android Malware: Characterization and Evolution*. http://www.ieee-security.org/TC/SP2012/papers/4681a095.pdf. (Cited on pages 9, 10, 11 and 84.)

[199] Zhou, Yajin, Zhi Wang, Wu Zhou, and Xuxian Jiang [2012]. *Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets*. http://www.csd.uoc.gr/~hy558/papers/mal_apps.pdf. (Cited on pages 14, 24 and 26.)

[200] Zimry, Irene, Raulf, Leong [2011]. *On Android threats Spyware:Android/SndApps.A and Trojan:Android/SmsSpy.D. - F-Secure Weblog*. http://www.f-secure.com/weblog/archives/00002202.html. Last visited on 2011-10-14. (Cited on page 10.)