ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Severin Amrein

Does your phone spy on you?

Master Thesis MA-2016-47
07.03.2016 to 05.09.2016

Tutor: Mirja Kühlewind
Co-Tutor: Brian Trammell, David Gugelmann
Supervisor: Prof. Laurent Vanbever

Abstract

More and more people store their sensitive data on mobile devices without really knowing who has access to this data. Apps must request permissions to access personal data. But as soon as the permission is granted, the user has no control anymore how the app uses the data and with whom the app shares it. Existing approaches try to detect such personal data leaks and try to support the user at making the right decision whether to continue using an app or to remove it. Unfortunately, most of these approaches have drawbacks such as requiring custom-built firmwares, rerouting the user's traffic to software middle-boxes or breaking the encryption, which puts the user's privacy and security at risk. This thesis presents an approach to detect personal data leaks that can be used by everyone with an iOS device without worrying about privacy or security risks. The thesis shows that it is possible to monitor all the traffic on an iOS device using Apple's network extension and to provide detailed information about the monitored traffic to the user about apps potentially leaking personal data to specific domains.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Motivation

In the past, personal computers have been the dominant device for processing and storing sensitive personal data. That is why many tools have been developed to protect sensitive data against being leaked to unwanted parties. Anti virus software informs the user after the detection of malicious activity on the computer. Plugins for all kind of browsers analyze a website's behavior in the background and prevent the user from being tracked. These days, however, a big amount of sensitive personal data is stored on mobile phones such as location data, photos, contacts or health data collected by the phone's sensors or by one of a growing selection of wearables. In contrast to the computer, a phone has far less security measures. It is even more critical because people want to share personal data but do not have much control over what happens. After giving an app access to the sensitive data, the user just has to trust, that the app treats the sensitive data with care and does not leak it to third parties. Personal data like the location, device information, contacts, calendar entries or login account information is in great demand in companies such as advertising companies or others to track the user or sell user data to other companies. Even more, because a phone accompanies its owner all the time what makes it possible to track users very precisely.
Table 1.1 presents existing approaches, which try to monitor the activity of applications installed on phones and try to enhance transparency regarding what personal data is sent and to where it is sent to. Enck et al. [1] developed a realtime privacy monitoring software called TaintDroid that is an extension to the Android platform. Using dynamic taint analysis, accessing and manipulating sensitive data from third party applications is tracked at the instruction level. However, this soluction can not be run as a simple app and needs to be installed using a custom-built firmware. Another approach, called PiOS, was presented by Egele et al. [2] in 2011. It is a tool to detect possible leaks in applications for Apple's iOS. Mach-O binaries, compiled from Objective-C code, are analyzed to point out data flows accessing personal data. Furthermore, there is a project called ProtectMyPrivacy (PMP) for jailbreaked iOS devices [3], which adds an additional layer between the applications and the operating system. This allows to intercept calls from the application to the official API's used to access personal data. That is why this approach allows the user to either grant or deny the access to personal data for an application. They also collect the user's protection decisions so that they can provide best suitable protection decisions to other users using crowd sourcing. There are also other approaches that intercept the applications's traffic instead of intercepting API calls within the operating system. One approach is from Ren et al. and is called ReCon [4], a cross platform system for Android, iOS and Windows phones. It reroutes a phone's traffic to software middle-boxes using VPN tunnels where machine learning helps to analyze the traffic. It detects personal data leaks in unencrypted traffic, visualizes with whom it is shared and also allows a user to customize what is shared. The project Haystack presented by Razaghpanah et al. [5] also monitors the traffic. Leveraging Android's VPN API, Haystack monitors all the traffic using a virtual interface. Unencrypted Traffic is analyzed and forwarded whereas encrypted traffic is intercepted by a SSL proxy. Therefore, they have access to all the traffic information in plaintext and can do a precise analysis regarding personal data leaks.
As pointed out in the above section, there are many approaches to analyze traffic and to detect

personal data leaks for Android phones. With 14.8% [6] of sales to the end user in 2016 Q1, iOS is the second most popular operating system behind Android. Therefore, we decided to develop an app running on iOS to detect personal data leaks based on network analysis.

Table 1.1: Existing approaches trying to monitor the traffic on mobile phones and to detect apps, which leak the user's sensitive data.

Project	Platform	Details	Drawback
TantDroid [1]	Android	Tracking at Instruction Level	Custom-Built Firmware
ReCon [4]	Multiplatform	Rerouting to Software Middle-Boxes	Only unecrypted Traffic, Analysis on Server
PiOS [2]	iOS	Mach-O Binary Analysis	Static Analysis
ProtectMyPrivacy [3]	iOS (Jailbreak)	Interception of API Calls	Jailbreak
Haystack [5]	Android	VPN Interface API, Analysis on the Phone	SSL Interception

1.2 Aim

We see drawbacks in all of the approaches for personal data leak detection mentioned in Section 1.1 (see Table 1.1). Our aim is to create a solution to detect apps leaking the user's personal data and to help the user deciding, which app should be deleted and which app can be used without any concerns. Our solution should be user friendly so that everyone with a iOS device can install it without harming any of Apple's policies and that it does not need any modification of the operating system like custom-built firmwares or jailbreaking. We also want to create an app that is not a privacy risk for the user. In contrary with rerouting the user's traffic to software middle-boxes for the analysis, we want to capture and process the traffic directly on the user's phone. Furthermore, we also think that privacy apps must not be a security risk and therefore, we want to show details about the traffic and data leaks without breaking encryption like Haystack [5] does with the SSL interception. SSL interception is very risky for the user. The user has to completely trust the SSL proxy and has no control about what happens between the proxy and the target domain. It could also be that sensitive data is stored in the proxy logs that can be revealed. Dormann calls attention in his blog about the risk of SSL interception [7]. He points out that it is very complicated to implement it in a completely secure way and that many applications performing SSL interception have flaws, which puts the user at risk. The validation of the upstream certificate could be incomplete or the SSL interception software could trust some malicious root CAs. Certainly, analyzing encrypted traffic (HTTPS) must be provided. Even more, because Apple will force developers to use encryption until the end of 2016 if they want to submit their apps to the App Store [8].

1.3 Approach

Our iOS app should be able to monitor incoming and outgoing traffic of all apps installed on the phone. Furthermore, it should show traffic statistics to the user such as which services are used and how much traffic it is sent using these services. The app should also provide details about domains where personal data is leaked to and also name apps, which have a high potential to leak personal data to these domains. In order to achieve these goals, we present the following approach consisting the two following phases.

App Screening Phase Because more and more apps encrypt their traffic, we need an approach that is able to classify apps sending encrypted and also unencrypted traffic. That is why

we introduce the App Screening Phase. This phase is about creating a setup to test apps regarding their potential to leak personal data of the user but only performed with test user data. This means that test users must not reveal any of their own personal data. Test users test several apps to generate traffic, which is intercepted using a SSL-capable proxy and captured on our server. This captured traffic is analyzed regarding personal data leaks using the test user data. The detected leaks are used in the Traffic Monitoring Phase to inform the user about potential leaks. This allows us to provide details about apps and potential leaks during the Traffic Monitoring Phase without rerouting the traffic or sending personal data of the test user to our server.

Traffic Monitoring Phase The Traffic Monitoring Phase includes monitoring all incoming and outgoing traffic on an iPhone using our own SpySpy app. In addition, the app uses the knowledge we obtain during the App Screening Phase to inform the user about apps that potentially leak personal data. The app also shows traffic statistics such as amount of traffic categorized by used services. All the information should be visualized clearly in order to help the user to understand what happens on the iPhone in the background and to help deciding, which app should not be used anymore.

1.4 Contributions

In this thesis, we present an approach to help the user deciding which apps can be used without concerns and which apps should better be deleted. Our SpySpy app monitors the user's traffic directly on the phone and informs the user about apps leaking personal data like location or device information and to which domains it is leaked.

1.5 Overview

This thesis starts with giving some background information about Apple's operating system iOS in Chapter 2. Afterwards, we describe the two phases of our approach. Firstly, the App Screening Phase is presented in Chapter 3 and secondly, the Traffic Monitoring Phase in Chapter 4, where we explain the design and implementation of the traffic monitoring app called SpySpy.

Chapter 2

Background

This Chapter aims to provide some background information about Apple's privacy policy in Section 2.1. Furthermore, we will point out some special characteristics of Apple's operating system iOS for mobile phones and tablets (see Section 2.2), which we faced during the app implementation. In Section 2.3 the NENetworkExtension framework is described, which provides APIs to customize and extend the core network features of iOS.

2.1 Apple and Privacy

The word privacy is heard a lot in connection with the brand Apple. That is because privacy has a really high priority for Apple. They try to make it very transparent how they protect you and your data and they are even one of the first companies that uses privacy as a sales tool. Tim Cook published a statement on Apple's website [9] where he makes clear how Apple thinks about their customers and privacy. He states they want to protect the customer's data with strong encryption and strict policies. He also makes clear that they never have and never will work together with government agencies from any country to let them access their servers or create a backdoor in Apple's software. Apple tries to communicate their view of privacy as clear as possible to their customers in order to build trust. But despite of their policies and their attempt to screen all apps before they can be downloaded in the Appstore, it is still not possible to ensure that no personal data of the user is leaked. Because apps need personal data like the location to provide their services, the user can give the permission to access the location. After giving the permission, it is completely up to the app what it wants to do with the user's location and with whom it wants to share it.

2.2 iOS

As can be read in the above Section 2.1, Apple tries to protect your data as good as they can. This section is about highlighting some of these measures in iOS. The operating system iOS was developed with security in mind. Users can not just disable security features like sandboxing or device encryption and other features like the touch ID make it very easy to secure your phone. But also these security measures make it challenging to implement something like our SpySpy app. We leverage the Network Extension API and use it in a way it was not supposed to be used. This section presents some of the features in iOS we will use or talk about in the following chapters. The information provided in this section is based on Apple's iOS Security White Paper [10].
In order to ensure that apps can not access files stored by other apps, iOS uses sandboxing to separate all apps from each other. This also means that an app can not modify any file from the operating system and has its unique home directory. It can only access files from other apps or share data with other apps by using official APIs provided by the operating system or using an extension provided by another app. Extensions can be provided by apps or the operating system to provide functionality to other apps. They can be restricted from only being used having the right entitlements. In order to use the Network Extension presented in Section 2.3, developers

have to apply for the entitlement on Apple's website. The application is then evaluated regarding how the developer want to use the extension. Obtaining the entitlement allows the developer to use the extension but only with the account he used for the application.
Apple provides a method to shared data between apps and extensions called App Groups. Developers can create App Groups within their account and can add apps and extension to shared data between them. However, only apps and extension owned by the same developer can be added to the App Group. After adding apps and extensions to an App Group, they obtain a shared on-disk container for storage. This container is created on the device as long as one member of the App Group is installed and is deleted as soon as every member is removed from the device. Furthermore, App Groups also allow to share preferences and keychain items.
Because apps are shielded from system files, files from other apps and because they run in a non-privileged user mode, it is impossible to run something requiring root privileges. This prevents us from using a raw socket in our SpySpy app to simply forward IP packets to the Internet. As an alternative, we have to parse IP packets and only forward the UDP or TCP payload using standard socket APIs, which are available from within the app (see Chapter 4). It also prevents that developers can access any device specific information. Neither the IMEI nor the UUID can be accessed [11]. The only identifier that is provided by the iOS is the Identifier for Advertising (IDFA). This identifier can also be controlled by the user. He can enable Limit Ad Tracking as written in Apple's privacy policy: "However, if you select Limit Ad Tracking on your mobile device, third party apps are not permitted by contract to use the Advertising Identifier, a non-personal device identifier to serve you targeted ads." [12].
Nonetheless, Apple makes its services as secure as possible for their users and ensure that no personal data is leaked to third parties, they also have to make sure that third party apps act just the same. That is why they introduced privacy control. This allows users to decide what data they want to share with an app and when the app is allowed to access it. Many apps use location services in order to obtain a user's location based on GPS, Bluetooth, crowd-sourced Wi-fi hotspot locations and cell tower locations. A user is asked to give the permission as soon as the app wants to get the location the first time. The user can even rethink this decision and change the permissions. Possible location service permissions are that the app is not allowed to use it, that the app can use it while running in the foreground or that the app can use it while running in the background. The same applies for personal data. As soon as an app wants to access some personal data like contacts, calendars, reminders or photos, the user is asked to give permissions and can also change it afterwards if desired.

2.3 NENetworkExtension

The NENetworkExtension framework was introduced during Apple's developer conference WWDC 2015. It provides different APIs, which allow developers to customize and extend the core network features of iOS and macOS. In order to use most of the classes provided by the framework, developers have to apply for the entitlement. The entitlement is mandatory to develop, test and release an app containing these classes. The following Section 2.3.1 is based on Apple's iOS Developer Library for the NENetworkExtension [13] and Section 2.3.2 is based on the iOS Developer Library for the NEPacketTunnelProvider. [14]

2.3.1 API Categories

The framework's APIs are organized in four categories. Each one of these categories contains families of APIs.

NEVPNManager NEVPNManager allows apps to create and manage personal VPN configurations for iOS and macOS in order to connect the phone with standard VPN servers. It is mostly used to provide services in order to protect a user's traffic during browsing or within public Wi-Fi networks.

NETunnelProvider NETunnelProvider allows apps to connect with non standard VPN servers. The app can implement the protocol of the client side itself. This category contains multiple APIs.

NEPacketTunnelProvider is the one we use to monitor the traffic and is explained in Section 2.3.2.

NEFilterProvider NEFilterProvider allows apps to filter the content. It can be used to protect students while they are browsing with their school devices.

NEHotspotHelper NEHotspotHelper allows apps to implement custom authentication to connect the device with Wi-Fi hotspots.

2.3.2 NEPacketTunnelProvider

NEPacketTunnelProvider is an API from the NETunnelProvider category. This API allows an app to implement a custom protocol for the client side in order to establish a VPN connection with a non standard VPN server. The NEPacketTunnelProvider can be configured and started from an app. After that, it runs as an extension in the background. This means that it keeps running even if the parent app is closed.
The extension sets up a virtual interface where traffic is routed through. Regarding the virtual interface, you can configure the virtual IP address, the DNS resolver, HTTP proxy, IP destination network to be routed through the tunnel and also the interface MTU. It also provides a NEPacketTunnelFlow object, which can be used to read from the network stack at the IP layer. The IP packets can then be encapsulated in a custom way and sent to the VPN server. The other way around works exactly the same. After receiving the packets from the VPN server, they are decapsulated and written back to the network stack, where they can be accessed by the original sender app.
Unfortunately, there is no possibility to map the packets coming from the network stack to the original sender app. Apple's developers responsible for the NENetworkExtension might provide this feature in the future.

Chapter 3

App Screening Phase

This Chapter is about the App Screening Phase of our approach to identify apps leaking personal data on an iPhone. Firstly, Section 3.1 describes the setup in order to test apps in a systematic way and capture their incoming and outgoing traffic. The second Section 3.2 presents the participants of the user studies and the apps that were tested. The last Section 3.3 describes the data set resulting from the two user studies. It also presents results such as different noteworthy findings occurred during the analysis that we want to emphasize, the feature analysis to detect personal data leaks based on analyzing the packet payload and also the feature analysis based on traffic metadata. The features based on the payload analysis of the data from the two user studies will then be used during the Traffic Monitoring Phase to provide detailed information about personal data leaks in the SpySpy app.

3.1 Setup

We want to create a setup, which allows us to test specific apps in a systematic way. This means that apps can be tested multiple times by multiple users using multiple iPhones. The configuration of the iPhone and the server should not require much time and test users have a clear process to test the apps. The setup should allow to add new iPhones, new test users and new apps in a convenient way. Furthermore, the setup must not leak any personal information of the test user but rather only use test user data, which allows us to detect leaks in the captured traffic of the apps. This can be achieved by only let test users test apps with preconfigured iPhones containing the special test user data.
In order to capture and analyze the traffic from a specific app, we need to reroute the traffic to an Ubuntu server, intercept it using a SSL-capable man-in-the-middle proxy and forward it to the target domain (see Figure 3.1). In the following Sections 3.1.1 and 3.1.2, components running on the iPhone and on the Ubuntu server are described separately. Section 3.1.3 describes the test process, Section 3.1.4 how to add new iPhones to the test setup and Section 3.1.5 how we analyzed the traffic after capturing.

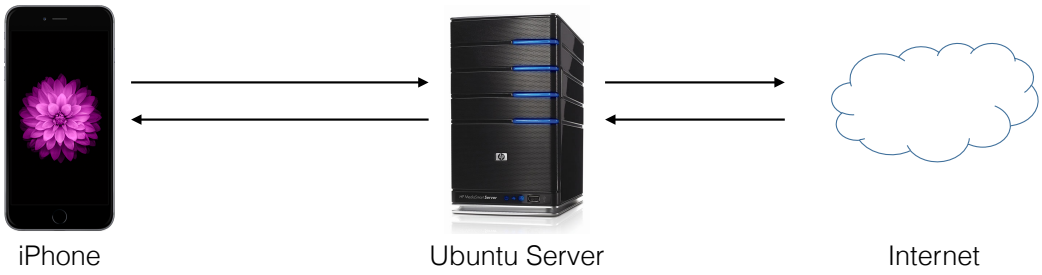

Figure 3.1: Setup to capture traffic caused by the tested app using a SSL-capable proxy on an Ubuntu server [16], which intercepts the traffic between the iPhone [15] and the target server.

3.1.1 iPhone Components

As you can see in Figure 3.1, all the traffic from the iPhone is rerouted to an Ubuntu server. We configured the iPhone to use a proxy running on the Ubuntu server. Therefore, the proxy's certificate has to be installed so that it is able to intercept HTTPS traffic. Because we use a proxy called Mitmproxy [17], the certificate can be installed very easily by opening the website 'http://mitm.it'. We also configure a VPN tunnel between the iPhone and the Ubuntu server to ensure that also other traffic than web traffic is rerouted to it such as ICMP. In order to simplify the configuration of the iPhone, we created a profile with the Apple Configurator 2 (Version 2.2.1) [18] and added the VPN configuration and the proxy configuration. This allows us to configure the iPhone installing only this profile.
The other part of the configuration is to make sure that we always use the same test user data to test an app. This allows us to detect leaks of personal data during the traffic analysis of the captured traffic. The iPhone is configured with a test user Apple ID and personal data like contacts, calendar entries, reminders and images is stored in the iCloud. These information can be downloaded to each iPhone very fast by only enabling iCloud. In order to install all the apps, we used iTunes logged in with the same Apple ID and added all apps to the Apple ID. Connecting the iPhone to iTunes allows to install the desired apps very convenient without searching every app in the Appstore on the iPhone. Furthermore, we created a Facebook and a Gmail account with the same test user data in order to log in to services of different apps.

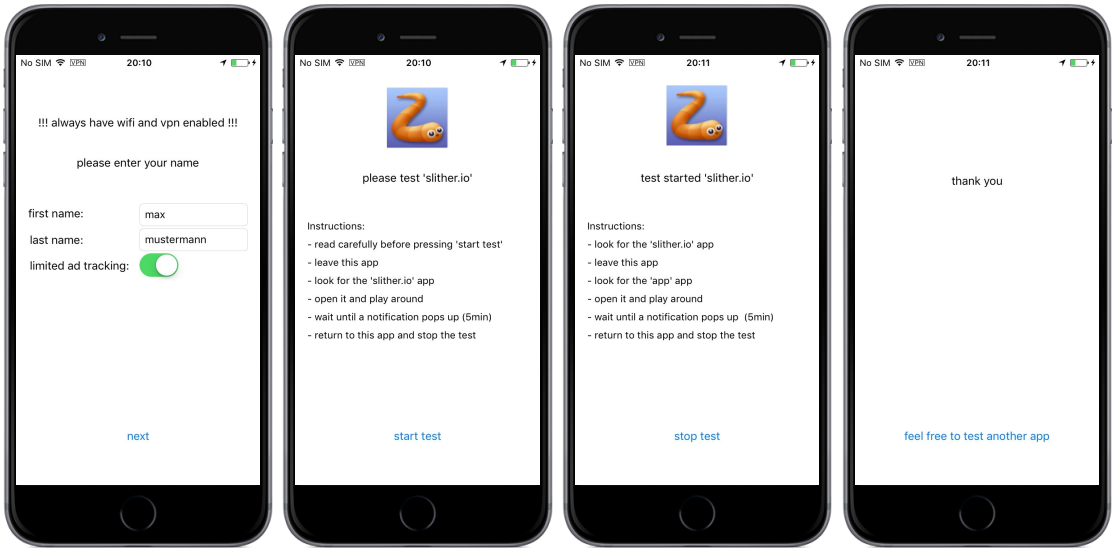

Figure 3.2: Screenshots of the App Testing app running on the iPhone. It communicates with the Ubuntu server in order to tell the user, which app should be tested and also to lead the user through the test process.

The last component is the App Testing app. Figure 3.2 shows screenshots of the App Testing app. It is responsible for the communication with the Ubuntu server in order to tell the user, which app should be tested and also to lead the test user through the test process. More details about how the app works can be found in Section 3.1.3, where we describe the test process.

3.1.2 Server Components

In order to reroute and intercept the iPhone's traffic to the Ubuntu server (Ubuntu 14.04.4 LTS), we need some components running on the server. The proxy software Mitmproxy [17] is a SSL-capable man-in-the-middle proxy software for HTTP. This software (Mitmproxy 0.18) can intercept HTTP requests and responses on the fly and can make scripted changes to them. It can also save and replay requests and responses and create SSL certificates for on the fly

interception. In order to let the Mitmproxy intercept all the encrypted HTTP traffic, the client device has to trust the proxy's built-in certificate authority (CA). After configuring the proxy on the device, visiting the domain 'mitm.it' lets you easily configure the proxy certificate on the device. [19]
Another component is a software called Strongswan (Linux strongSwan U5.1.2/K4.2.0-41-generic). It is used as a VPN server so that the iPhone can establish a VPN tunnel to the Ubuntu server. This software is a multiplatform OpenSource IPsec implementation based on the FreeS/WAN project and has been developed by the Institute for Internet Technologies and Applications at the University of Applied Sciences Rapperswil in Switzerland. It implements both the IKEv1 and IKEv2 protocols and focuses on simple configuration, strong encryption and authentication methods, powerful IPsec policies supporting large and complex VPN networks and modular design with great expandability. We used the IKEv2 protocol to establish the VPN tunnels. [20] [21]
A MYSQL database runs on the server in order to track the test progress. The database contains three tables. The 'apps' table contains all apps that should be tested, the last time it was tested and a counter to track how many times the app has been tested. The 'users' table stores all test users, their names and the date when the user tested an app the first time. The third table called 'apps_tested_by_user' uses information of the 'apps' and 'users' table to track the test progress and creates an entry for each pair of tested apps and test users with the corresponding date. This table is also used for tracking which app's captured traffic has already been processed and analyzed. Once a day, a cron job looks for entries in this table, which have not been processed yet, and starts to analyze the corresponding traffic.
The last components is a server software written in Python to communicate with the App Testing app on the iPhone. Messages are sent using TCP for the communication between the App Testing app and the server software. As soon as the iPhone app requests an app to test, the server selects an app based on information stored in the database and sends it back to the App Testing app. It checks if there is an established VPN tunnel or if already all apps are tested by a particular user. This software also starts the Mitmproxy and two tcpdump instances to capture the traffic. One for the traffic going through the proxy and one for the traffic, which is forwarded by the Ubuntu server without going through the proxy. Therefore, all traffic is captured in different ways to be sure to save as much information as possible about the traffic. The Mitmproxy and the tcpdump instances have to be started for each app again in order to ensure files containing captured traffic for each tested app and user. Finally, we end up with the following five files containing information about the traffic caused by the tested app:

<AppName>-<UserFirstName>-<UserLastName>-deviceinfo.txt This file saves device information received from the App Testing app running on the iPhone such as if Limit Ad Tracking is enabled on the iPhone, the iPhone's location or specific device information like battery state or operating system version. Furthermore, it also contains the carrier name, IP address and the iPhone's current Identifier for Advertising (IDFA, see Section 2.2). The information in this file is used during the traffic analysis to look for leaks in the request payload.

<AppName>-<UserFirstName>-<UserLastName>.txt This file saves the output of the Mitmproxy. The output contains error messages for failed handshakes and request/response summaries.

<AppName>-<UserFirstName>-<UserLastName>.dump This file saves all the traffic in a specific Mitmproxy format. It can be used to replay the traffic. That is why this file is used to analyze the traffic.

<AppName>-<UserFirstName>-<UserLastName>-proxy.pcap This is a pcap file recording the traffic that goes through the proxy using tcpdump.

<AppName>-<UserFirstName>-<UserLastName>-direct.pcap This is a pcap file recording the traffic that does not go through the proxy using tcpdump like ICMP packets for example.

3.1.3 Test Process

This section describes how a user tests an app on the iPhone using our setup explained before and how the different components work together to capture the traffic. Figure 3.2 shows the App Testing app running on the test iPhone, which leads the user through the test process and communicates with the server software. Screenshot 1 lets the user enter first name, last name and also if Limit Ad Tracking is enabled on this iPhone. After hitting the 'next' button, the App Testing app checks for a running server software and signals the server software that there is a user who wants to test an app. The server software checks for a running VPN tunnel between the iPhone and the server. If everything is fine, the server software selects an app based on information from the database and sends it back to the app. The App Testing app present it to the user besides instructions how the user has to test the app (see screenshot 2). Hitting the 'start test' button, the user signals the server that the test is started and also sends device specific information like the user's location to the server. The server software stores the device information, starts an Mimtproxy instance listening on the port communicated by the App Testing app and also starts the two tcpdump instances (see Section 3.1.2). The device information file and the output files of the Mitmproxy and the tcpdumps are stored in a temporary directory. After starting the test, the user leaves the App Testing app and opens the specific app. A notification will inform the user after 5 minutes that the specific app can be closed. Going back to the App Testing app, a 'stop test' button appears, which the user can hit in order to stop the test (see screenshot 3). Hitting this button will also signal the server software that the user has stopped testing. The server software stops the Mitmproxy and the two tcpdump instances and moves the corresponding files from the temporary directory to a final directory. If something has been going wrong during the test, the user closes the App Testing app or the iPhone has been shut down, the files are not moved to the final directory. Files that stay in the temporary directory are deleted every night by a cron job. The server software updates the database if the test was successfully completed. Otherwise the test user has to test the app again. The last screenshot in Figure 3.2 thanks the user. If users want to test another app, the 'feel free to test another app' button will take them to screenshot 2 presenting another app that has to be tested.

3.1.4 How to add a new phone

This setup is developed with scalability in mind. That is why it is made very easy to add new apps to test and also new phones, so that more users can test at the same time. The following things have to be considered in order to add a new app or a new iPhone.

New iPhone In order to add a new iPhone, it has to run at least iOS 9. Next, the profile created with the Apple Configurator 2 is installed on the iPhone. Every test iPhone has its own port to connect to the Mitmproxy and to the appropriate VPN server configuration. If the port 8080 is used in the VPN configuration profile, you also have to adjust the port in the App Testing app source code and the VPN configuration on the Ubuntu server. This makes sure that multiple phones can test apps simultaneously.

New App Adding a new app is very trivial. Just add it to the 'apps' table in the database on the Ubuntu server. It is also possible to specifically define apps in the traffic analysis server software for the case you added apps to the database but do not want them to be tested by the users.

3.1.5 Traffic Analysis

The used Mitmproxy to capture traffic allows to read and dump captured web traffic from a file in Mitmproxy format. Therefore, each file containing captured traffic from each app and test user can be read using Mitmproxy. Each request is parsed for personal data and the result is saved in a CSV file. The personal data is taken from the test user data that we store in the test user's Apple ID iCloud and also from the appropriate device information file. We also save other details about requests and responses in the CSV file such as scheme (HTTP or HTTPS), method (GET, POST, etc.), port, path, HTTP version and timestamp. Additionaly, also HTTP header information like content type, accept-encoding, accept-language and user agent. Domains are cropped in

order to obtain only effective second level domains (eSLD). These eSLDs are compared with the Ad Block Plus domain blacklist. If they are found on the blacklist, they are flagged as blacklisted domains. The effective second level domains are also compared to the app names. If the eSLD contains the app name, it is flagged as the app's own domain. Otherwise, it is flagged as a third party domain. As mentioned before, the traffic analysis creates a CSV file for each parsed Mitmdump file and also a summary file including all individual CSV files. The CSV file can be used to analyse the results further for example by using Python's Pandas library [22], which is very suitable for big correlations. During the traffic analysis, we also filter the traffic captured by the Mitmproxy based on source IP addresses. Unfortunately, we used the Mimtproxy without any password protection in the beginning. Therefore, other parties also used the Mitmproxy at some point and compromised our captured traffic. Because users performed the testing only at a few locations, we were able to identify the used source IP addresses and to filter the captured traffic.

3.2 User Studies

We want to analyze traffic, which is very similar to real app traffic of an iPhone in order to end up with a meaningful statement. That's why two user studies were performed for testing different apps on the iPhone. The apps were tested on iPhones at least Model 4S running iOS 9 and higher.
The first user study (user study 1) aims to simulate real user traffic by testing very popular apps. We decided to take the top 50 apps of the Switzerland iTunes charts for free apps (date: 2. May 2016) and added some apps with a high potential to leak personal data (see Appendix A.1). Apps with a high potential are dating apps (Tinder and Lovoo), news apps (Tagesanzeiger, 20min and Watson), a live TV app (Zatoo Live TV), games (Fruit Ninja and Despicable Me), a flashlight app (Flashlight) and a social media app (Linkedin). All these apps provide their service for free and are suspected to collect as much user data as possible. The five test users participating in this user study presented in Table 3.1 have different occupations and are from different age groups. They tested the apps for 5 minutes on prepared iPhones. User five tested when the apps were first opened right after the installation and therefore asked for the user's permission to access personal data. We assume that apps send information as soon as they have access to it. The other four users tested the apps after the permission was already granted.
A second user study (user study 2) was performed the same way the first one was but concerning location leaks. Therefore, we choose different apps than in user study 1 such as news apps, weather apps, games, travel apps, dating apps or travel apps, which have a very high potential to leak the user's location (see Appendix A.2). This user study was only performed by user 5 (the author of this thesis) and each app was tested right after installation when the apps ask for the permission to access the user's location.

Table 3.1: Details about the test users, who participated in the two user studies to test apps potentially leaking personal data.

No.	Age	Occupation	Further Information
1	25	Business Manager	-
2	21	Student	-
3	49	Anesthesist	-
4	24	Engineer	-
5	24	Engineer	Apps tested right after the installation when they ask for the permission to access the user's personal data

3.3 Results

This section first describes the data collected from the two user studies (see Section 3.3.1). Section 3.3.2 presents noteworthy findings we want to point out and Section 3.3.3 presents the result of the feature analysis based on parsing the payload for personal user data. Section 3.3.4 explains the traffic analysis based only on metadata of the traffic. All the analysis in this section is performed based on captured HTTP and HTTPS traffic.

3.3.1 Data Set

During the testing process, we noted that not all of the apps work properly if their traffic is intercepted by a SSL-capable proxy. Further investigation indicated that the connection was always aborted. An explanation for this behavior is certificate pinning, which verifies the certificates. Therefore, we were not able to analyze the traffic of these apps. 13 of the 60 apps tested in user study 1 (see Appendix A.1) and 17 of the 98 apps tested in user study 2 do not work properly (see Appendix A.2). Most of the apps, which do not work as expected, are apps from big service vendors like Google or Facebook.
Table 3.2 shows the number of captured HTTP request and response pairs. In the following, we use the term request to refer to request and response pairs. 16473 HTTP and 23021 HTTPS requests were captured during user study 1 and 8867 HTTP and 9570 HTTPS requests during user study 2. This results in 25340 HTTP and 32591 HTTPS requests in total. Around 56.25 %, more than half of the requests, is encrypted using HTTPS. We assume that this percentage will increase quickly in the future, because Apple will force app developers to use HTTPS in order to get accepted to provide their apps in the Appstore (see Section 1.2).

Table 3.2: Description of the data set, which is the result of the two user studies. It shows the number of captured requests.

	HTTP	HTTPS
User Study 1	16473	23021
User Study 2	8867	9570
Sum	25340 (43.75 %)	32591 (56.25%)

3.3.2 Noteworthy Findings

During our traffic analysis, we encountered very interesting app behaviors. This section presents some apps, which act in a way we would not have expected.
The first app we want to examine in detail is the 'SBB Mobile' app. The company SBB is financed partly by the government and their interest should not be to generate as much money as possible using their customers as a product. But nonetheless, the SBB Mobile app sends their user's location to a domain called 'smartadserver.com', like other Swiss apps for example '20 min' or 'Watson'. SBB sends the location even though they do not show any advertisement in the app. It seems likely that they provide user information to this domain in order to benefit from a service provided by this domain. Unfortunately, we could not verify this behavior in the new SBB app called 'SBB Mobile Preview' because it does not work with our setup using the SSL-capable proxy due to certificate pinning.
The second app is a social media app for business people called 'Linkedin'. This is the only app for which we detected the upload of contacts and calendar entries. We have to point out that the user is informed very well before the app asks for the permissions to access the personal data. Linkedin states that the app uploads the contacts and calendar entries from time to time in order to improve their services.
The last finding is the most interesting and also the most alarming one. The app 'Magellan Active' of the company Magellan sends the login information in clear text (HTTP). This app provides services to improve your health like support during running with GPS tracking. It also tracks your daily activity such as steps, calories, distance and sleep information. With sniffing

the traffic of this app during the login process, a third party can get access to the profile and the user's health data. [23]

3.3.3 Feature Analysis: Payload

This section is about the feature analysis based on analyzing the packet payload. Features are the different data leak categories, which are used later to show details about domains in the SpySpy app (see Chapter 4). Using a SSL-capable proxy gives us access to the data sent by the tested apps by looking at the packet payloads. Therefore, we are able to parse the HTTP and HTTPS requests and identify data leaks. We can do that because we set up specific test iPhones with specific data (see Section 3.1). This allows us to parse the traffic for this specific data. This section shows the different categories of leaks, which are differed by the type of leaked data. The goal is to detect which apps leak what data and also to which target domain they leak it. We differ between leaking data to the app's own domain and leaking data to a third party domain. Because it is not trivial to separate domains into own and third party domains, we applied the following definition. If the name of the app is part of the effective second level domain, it is a own domain. Otherwise we rate it as a third party domain. Furthermore, we also have to state that it is possible that apps encrypt or modify their sent data additionally to the SSL/TLS encryption of HTTPS so that we are not able to detect data leaks. The following paragraphs show the different categories and their definitions.

Location Leaks Location leaks are defined by detecting the user's location in the outgoing traffic of the iPhone. The longitude and the latitude are cropped to two decimal places. We look for the longitude and the latitude separately and use either a dot or a comma to separate the left and the right part of the floating point number.

Identifier for Advertising (IDFA) Leaks The Identifier for Advertising (IDFA) is a ID provided by Apple's iOS in order to let the developers uniquely identify the device without giving them access to device specific IDs like the IMEI or the MAC address. As explained in Section 2.2, Apple's policy does not allow to use this ID if the user enables the Limit Ad Tracking on the iPhone. Therefore, we defined IDFA leaks as detected IDFAs even if Limit Ad Tracking is enabled on the iPhone.

Personal Information Leaks Personal information is information of the following four categories, which apps can access on the iPhone after obtaining the appropriate permission. The categories are contacts, calendars, reminders and photos. If an app sends data of one of these categories, we take it into account as a Personal Information Leak.

Login Data Leaks In order to use the services provided by some apps, the user has to log in with a Facebook or a Gmail account or has to create an account for the specific service provided by this app. The accounts are always created with the same user credentials. Therefore, we are able to identify these leaks in the captured network traffic using these credentials.

Device Information Leaks Device information is information about a specific device. We parse for the following information:

- **Name**: The name of the iPhone.
  Example: John Doe's iPhone
- **System Name**: The operating system running on the iPhone.
  Example: iPhone OS
- **System Version**: The version of the operating system running on the iPhone.
  Example: 9.x.x
- **Model**: Hardware model of the iPhone regarding the hardware.
  Example: iPhone6,2

- **Carrier Name**: Name of the carrier used by the user.
  Example: Sunrise
- **Battery State**: State of the battery of the iPhone.
  Example: unplugged
- **Battery Level**: Battery level of the iPhone from 0 to 1.
  Example: 0.77
- **IP**: The local IP address of the device.
  Example: 192.168.1.100

Table 3.3 and 3.4 show the results of the feature analysis of parsing the packet payload using the capture traffic of the two user studies. It shows that there are only two apps in user study 1, which leak personal data. One is the 'Gmail' app that uses the contacts to update your Gmail contact list and the other one is 'Linkedin' already mentioned in Section 3.3.2. It is surprising and also pleasing that only two apps of the in total 158 apps leak personal data to their own domain and no app leaks personal data to a third party domain. Regarding the Location Leak category, we obtained a result that we were expecting. Many apps use the user's location in order to provide their services like navigation or location based information like weather or dating apps. Because they need the location to provide their core service, they can be sure that the user will probably give the app the permission to access the location. Therefore, they can also use the user's location to benefit from third party services, which require the user's location. Both user studies show that almost 20 % of the tested apps send the location to their own domains. However, 62 % in user study 1 and 29 % in user study 2 also leak the user's location to a third party domain. The next leak category is the IDFA Leaks category. As the name says, Identifier for Advertising is supposed to be used for advertising. Apps often use third party services to provide advertisements to the user. Both user studies exactly confirm this assumption. Only 5 % of the apps use the IDFA for their own purposes but 50 % and more apps leak the IDFA to third party domains even if the user does not agree with using the IDFA by enabling the Limit Ad Tracking on the iPhone. The Login Data Leaks category has to be treated with care. As described earlier, this category includes leaks of account information of Facebook, Gmail or app specific accounts. Therefore, it is explainable that apps send the login data to their own domains. The leaks to third party domains have to be explained. Many apps use Facebook services to let the user log in using their Facebook account. That is why the apps have to send the user's Facebook name to one of Facebook's domains what leads to higher number of apps for the Login Data Leaks category leaking to third party domains. The last category Device Information Leaks is a very interesting one. Table 3.3 and Table 3.4 show that many apps send device information to their own domains (75 % and 49 %). Even more interesting is that with 92 % and 90 %, most of the apps send the device information to a third party domain. The question is why device information is needed by all these domains. We assume that some use it to provide the right service to the user for example to distinguish between different operating systems or device model. However, we are also convinced that some services like advertising or tracking services just try to collect as much user specific information as possible. Because information can be sold to other companies, they try to use their customers as a product. Altogether, both tables show quite similar results on all of the leak categories and also show that many apps leak the user's personal data to third party domains.
Because of this traffic analysis based on parsing the payload, we are able to provide detailed information about what data is leaked to which domain by which app. Taking the data from both user studies and performing this analysis allows to inform the user about apps leaking personal data tested during the two user studies.

Category	Target Domains	# Apps (%)	App Names
Location Leaks	3rd Parties	37 (62.0%)	uber, shazam, musikplayer, despicableme, jodel, linkedin, youtube, itunesu, colorswitch, tripadvisorhotels, stack, instagram, spotify, googleuebersetzer, clashroyale, riskyroad, magicpiano, 20min, tinder, whatsapp, meteoswiss, facebookmessenger, tagesanzeiger, freimusikplayer, facebook, googlemaps, leagueofwar, sbbmobile, dragonhills, kostenlosmusik, wish, schweizerfleisch, watson, viber, lovoo, musical.ly, dreamleaguesoccer
Location Leaks	Own	10 (17.0%)	uber, shazam, booking.com, tripadvisorhotels, meteoswiss, tagesanzeiger, lovoo, 20min, tinder, sbbmobilepreview
IDFA Leaks	3rd Party	37 (62.0%)	uber, visagelab, shazam, musikplayer, jodel, zattoolivetv, pianotiles2, youtube, colorswitch, tripadvisorhotels, stack, zalando, instagram, flashlight, spotify, pinterest, imusices, leagueofwar, magicpiano, akinatorthegenie, 20min, tinder, beautyplus, gmail, tagesanzeiger, footballstar2016, fruitninja, riskyroad, sbbmobile, kostenlosmusik, wish, schweizerfleisch, watson, viber, lovoo, musical.ly, dreamleaguesoccer
IDFA Leaks	Own	3 (5.0%)	flashlight, akinatorthegenie, sbbmobilepreview
Personal Data Leaks	3rd Party	0 (0.0%)	-
Personal Data Leaks	Own	2 (3.0%)	linkedin, gmail
Login Data Leaks	3rd Party	13 (22.0%)	uber, spotify, kostenlosmusik, musikplayer, wish, youtube, watson, magicpiano, lovoo, tripadvisorhotels, tinder, gmail, sbbmobilepreview
Login Data Leaks	Own	10 (17.0%)	uber, shazam, instagram, booking.com, skype, linkedin, tinder, musical.ly, zalando, gmail
Device Information Leaks	3rd Party	55 (92.0%)	uber, visagelab, whatsapp, shazam, instagram, despicableme, jodel, tripadvisorhotels, zattoolivetv, linkedin, pianotiles2, itunesu, slither.io, youtube, colorswitch, skype, stack, zalando, gmail, flashlight, spotify, googleuebersetzer, booking.com, clashroyale, magicpiano, imusices, akinatorthegenie, 20min, tinder, beautyplus, slither.ioR, riskyroad, meteoswiss, facebookmessenger, tagesanzeiger, freimusikplayer, facebook, snapchat, googlemaps, fruitninja, leagueofwar, sbbmobile, dragonhills, kostenlosmusik, wish, pinterest, schweizerfleisch, watson, viber, lovoo, musikplayer, musical.ly, footballstar2016, dreamleaguesoccer, sbbmobilepreview
Device Information Leaks	Own	45 (75.0%)	uber, visagelab, shazam, instagram, jodel, tripadvisorhotels, zattoolivetv, linkedin, pianotiles2, slither.io, colorswitch, skype, zalando, gmail, flashlight, spotify, googleuebersetzer, booking.com, imusices, magicpiano, akinatorthegenie, 20min, tinder, beautyplus, slither.ioR, meteoswiss, facebookmessenger, tagesanzeiger, freimusikplayer, facebook, snapchat, googlemaps, fruitninja, whatsapp, sbbmobile, wish, pinterest, youtube, watson, viber, lovoo, musikplayer, musical.ly, footballstar2016, sbbmobilepreview

Table 3.3: This table shows apps, which leak data to their own or third party domains based on the captured traffic from user study 1 divided in leak categories. The table shows the absolute number of apps, the percentage and also the app names.

Category	Target Domains	# Apps (%)	App Names
Location Leaks	3rd Parties	28 (29.0%)	ingress, cyclemeter, cheapflightshere, sickweather, turf.ly, runmetergps, secretescape, imatechinese, layar, silicon.de, snapguide, untapped, topomaps, forecast, swarm, foursquare, triposo, scoutlite, landlord2, speedtest.pro, orbitz, speedtest, tuneln, scout, ebookers, geocaching, zaploot, yahooweather
Location Leaks	Own	18 (18.0%)	yahooweather, citymapper, orbitz, ebookers, sickweather, turfwars, zaploot, turf.ly, speedtest, airbnb, runtastic, foursquare, speedsmartwifi, tracker.com, triposo, heremaps, priceline.com, meteoblue
IDFA Leaks	3rd Party	49 (50.0%)	ricardo.ch, vivino, cyclemeter, sickweather, turf.ly, sygic, strava, runmetergps, slickdeals, silicon.de, yahooweather, wanderbock, jetradar, snapguide, autoindex, untapped, topomaps, turfwars, forecast, airbnb, swarm, flights, foursquare, triposo, stucard, runkeeper, runnersfree, landlord2, speedtest.pro, orbitz, raincoat, homegate.ch, speedtest, tuneln, runtastic, scout, tripit, waze, swisssnow, freegps, ebookers, geocaching, navmiiswitzerland, whereamiat, citymapper, glympse, heremaps, home.ch, stepz
IDFA Leaks	Own	5 (5.0%)	airbnb, propertycomparis, speedtest, priceline.com, forecast
Personal Data Leaks	3rd Party	0 (0.0%)	-
Personal Data Leaks	Own	0 (0.0%)	-
Login Data Leaks	3rd Party	19 (19.0%)	runnersfree, landlord2, strava, snapguide, autoindex, sickweather, geocaching, turf.ly, runtastic, flights, foursquare, tripit, uepaa, slickdeals, secretescape, inroute, runkeeper, untapped, feedly
Login Data Leaks	Own	9 (9.0%)	snapguide, turf.ly, zaploot, airbnb, swarm, runtastic, tripit, magellanactive, nikerunning
Device Information Leaks	3rd Party	88 (90.0%)	ingress, cyclemeter, cheapflightshere, turf.ly, whiterisk, uepaa, forecast, swisssnow, autoindex, amaze, flights, foursquare, navmiiswitzerland, runnersfree, speedtest.pro, homegate.ch, runtastic, scout, parallelmafia, tomtomgo, easyjet.mobile, stepz, ricardo.ch, landlord2, trails, speedsmartwifi, layar, untapped, imatechinese, airbnb, mobilenetworksunrise, nikerunning, propertycomparis, sickweather, landiwetter, gpslogbooks, raincoat, tripit, galileo, meteoblue, inroute, zombiesrun, gctools, whereamiat, citymapper, glympse, palnet105, home.ch, priceline.com, trackslogger, sygic, magellanactive, silicon.de, wanderbock, snapguide, swarm, scoutlite, runkeeper, simplegps, waze, turfwars, mytracks, feedly, freegps, ebookers, geocaching, heremaps, onthefly, myswisscom, vivino, strava, runmetergps, slickdeals, secretescape, jetradar, tolllookoutgps, topomaps, inrixtraffic, triposo, whereami, stucard, orbitz, speedtest, tuneln, dropbox, zaploot, yahooweather, myupc
Device Information Leaks	Own	48 (49.0%)	turf.ly, ricardo.ch, tourality, vivino, myswisscom, trails, slickdeals, whiterisk, sygic, waze, runmetergps, magellanactive, secretescape, layar, yahooweather, feedly, jetradar, snapguide, zombiesrun, inrixtraffic, forecast, airbnb, swarm, flights, foursquare, mobilenetworksunrise, nikerunning, easyjet.mobile, turfwars, strava, homegate.ch, speedtest, runtastic, tripit, triposo, meteoblue, stucard, freegps, gctools, zaploot, citymapper, myupc, heremaps, palnet105, home.ch, propertycomparis, priceline.com, speedsmartwifi

Table 3.4: This table shows apps, which leak data to their own or third party domains based on the captured traffic from user study 2 divided in leak categories. The table shows the absolute number of apps, the percentage and also the app names.

3.3.4 Feature Analysis: Metadata

We also analyzed the detected leaks from Section 3.3.3 regarding packet metadata. Firstly, we assign the leaks to tuples regarding the tested app, the test user and the target domain. Such a tuple would be for example 'app-testuserfirstname-testuserlastname-domain'. After analyzing the location leak timeline of different tuples, we wondered if there is a correlation between leaks and packet metadata. Figure 3.3 shows the location leak timeline of several tuples. The legend shows the tuples plus an ID, which corresponds with the y-axis. Looking at the tuple with ID 3, you can see that the location is leaked every 10 to 12 seconds. This lets us assume that some of the tuples send the location in a periodic manner and that third party services like tracking services request the user's location periodically. However, other tuples like the tuple with ID 1 look more like some apps send the user's location randomly or triggered by some user actions. Another assumption is that some services request user data always in the same format such as the latitude and longitude of the user location. This would lead to requests with same size, meaning that leaks could be detected by measuring the request size.

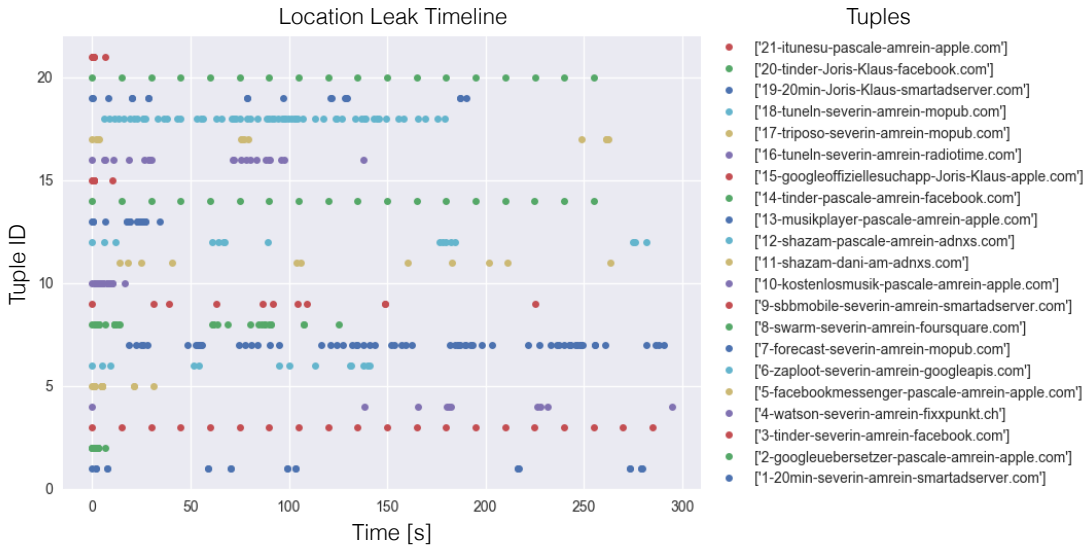

Figure 3.3: Location leak timelines of different apps to specific domains tested by test users. The tuples in the legend contain also an ID, which can be use to identify the tuple in the graph (y-axis).

The following list shows all analyzed metadata features. We analyzed the features for each of the different categories from Section 3.3.3 separately. Some categories might be detectable using metadata features and others might not. We also investigate all possible combinations of two metadata features. There might be correlations between two of the features.

- **Timestamp Frequency**: This feature analyzes if personal data is leaked in a periodic way by using autocorrelation on the timestamps of the leaks. We only calculated the this feature for tuples with more than 5 request per test.
- **Updload/Download Ratio**: This feature is about the ratio of the request and the response size. We calculate the ratio of each request and response pair and average them. But we also first average the size of all requests and responses before calculating the ratio.
- **Request Mean Size**: Mean of all sent requests leaking personal data.
- **Domains per App**: Analyzing the number of domains an app leaks personal data to.
- **Apps per Domain**: Analyzing the number of apps that leak personal data to a domain.

Unfortunately, the result of the metadata feature analysis showed that metadata features can not be used to distinguish tuples leaking a user's personal data and tuples not leaking personal

data. Nevertheless, we present an example in this section in order to show how we did the analysis and how we concluded that these features can not be used for personal data leak detection. The examples show the Timestamp Frequency feature and the Packet Mean Size feature analyzed regarding the Location Leak category. This means that we tried to distinguish tuples leaking the user's location from tuples that do not leak the user's location using these two metadata features.
Single features are analyzed using kernel density estimation (KDE). We used the Python module called Seaborn, which provides a high-level interface for drawing attractive statistical graphics [24]. KDE estimates the probability density function of a random variable in a non-parametric way and is closely related to histograms. The histogram considers the random variable within the bins and sums it up, where as the kernel density estimation uses Gauss for each and sums up the gaussian functions [25]. We create a kernel density estimation for all tuples leaking the location and one for all tuples that do not leak the location in order to shows if a feature can be used to distinguish tuples that leak the user's location from tuples that do not leak the user's location. Figure 3.4 presents the kernel density estimations regarding the Timestamp Frequency feature of tuples that leak the user's location. Unfortunately, using the Timestamp Frequency feature does not let us distinguish between tuples with and without location leaks. Both densities look too similar. The same result showed up after analyzing the Request Mean Size feature (see Figure 3.5).
The already mentioned combination of two metadata features is shown in Figure 3.6 for tuples leaking the location and in Figure 3.7 for tuples not leaking the location. Figure 3.6 shows tuples with the Timestamp Frequency feature on the x-axis and the Request Mean Size feature on the y-axis. Most tuples have a small request mean size and also a small timestamp frequency so that the concentration of points increases toward the zero point. The same behavior can also be observed for tuples, which do not leak the location (see Figure 3.7). The point distribution in both figures is similar. Therefore, also combining these two metadata features does not allow to distinguish between tuples with and without leaks.
Summing up, single metadata features and also combinations of them can not be used to distinguish between tuples that leak the user's location and tuples that do not. The same applies for all other leak categories.

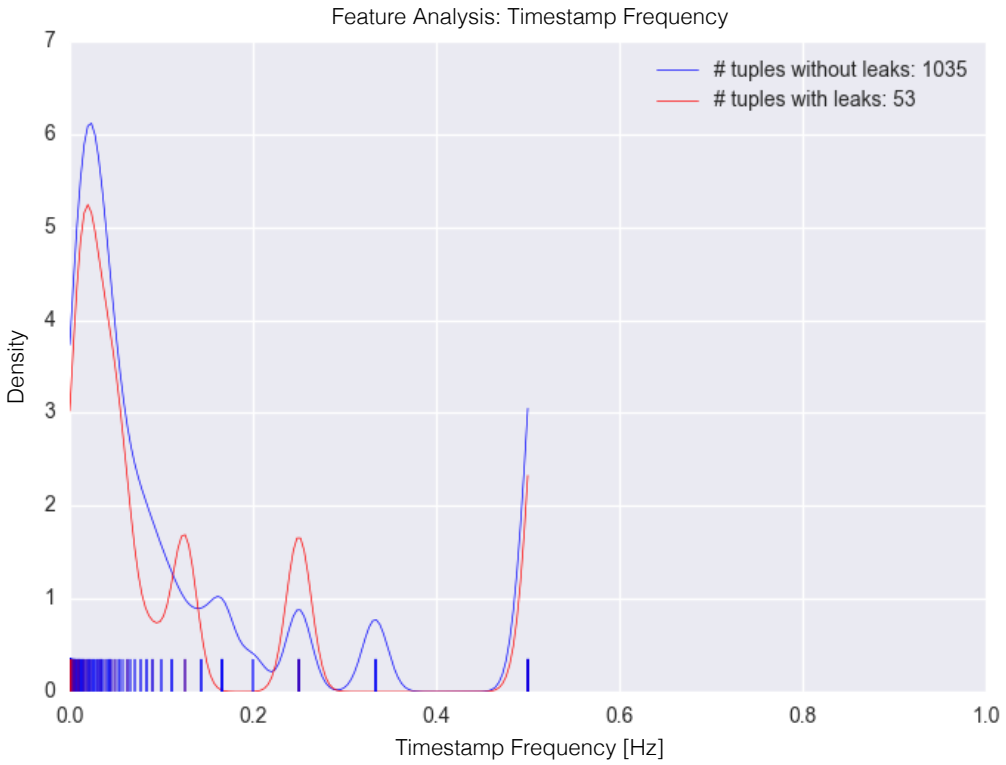

Figure 3.4: Kernel density estimation of the Timestamp Frequency feature of tuples. The red KDE is based on tuples containing requests that leak the user's location and the blue KDE is based on tuples containing requests that do not leak the user's location

Figure 3.5: Kernel density estimation of the Packet Mean Size leak feature of tuples. The red KDE is based on tuples containing requests that leak the user's location and the blue KDE is based on tuples containing requests that do not leak the user's location

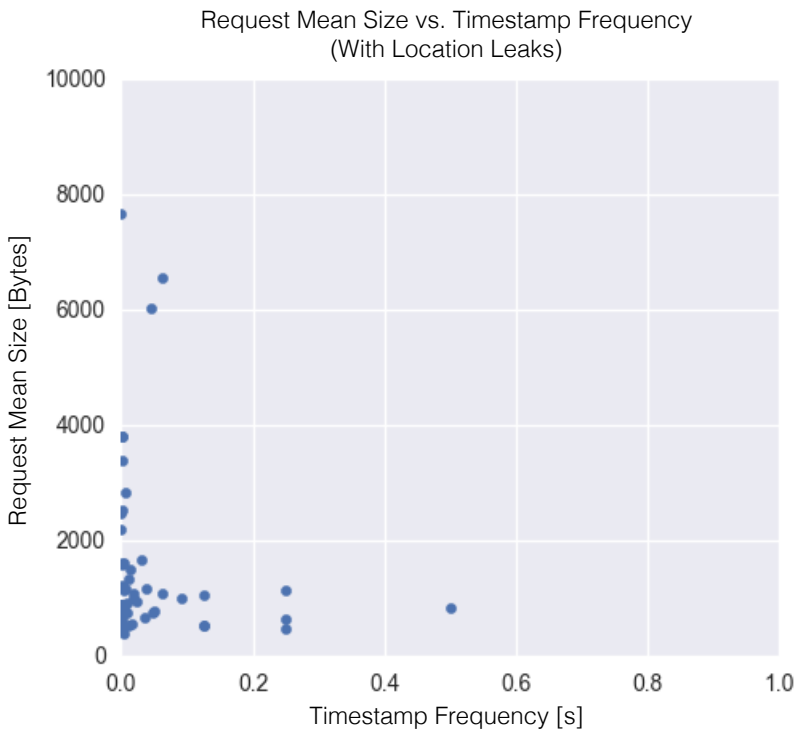

Figure 3.6: Timestmap Frequency feature together with the Request Mean Size feature of 53 tuples that leak the user's location.

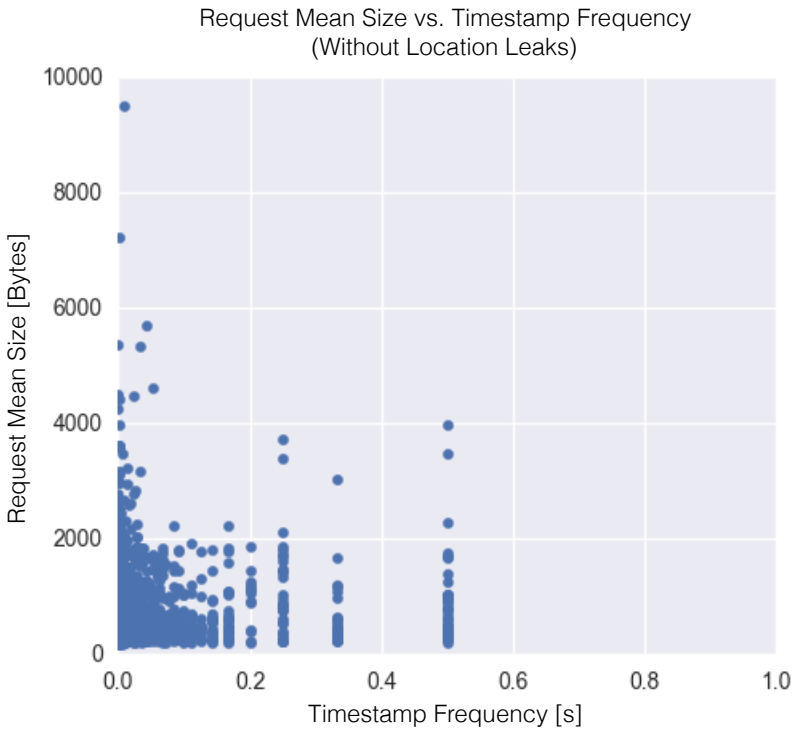

Figure 3.7: Timestmap Frequency feature together with the Request Mean Size feature of 1035 tuples that do not leak the user's location.

Chapter 4

Traffic Monitoring Phase

This chapter presents the Traffic Monitoring Phase, which is about implementing the SpySpy app in order to monitor and analyze the traffic locally on an iPhone and to inform the user about potential personal data leaks. Section 4.1 explains the design decisions and challenges that appeared during the app implementation. The second Section 4.2 describes how we tested the SpySpy app in order to ensure that the traffic forwarding works properly. Section 4.3 introduces the SpySpy GUI, its structure and visualization.

4.1 Design

This section presents the design of the SpySpy app. The app is built using different Apple and third party frameworks. The goal is to monitor all incoming and outgoing traffic of the installed apps, to analyze this traffic regarding personal data leaks locally on the iPhone and to visualize the results for users in order to help them deciding to use an app or to delete it.

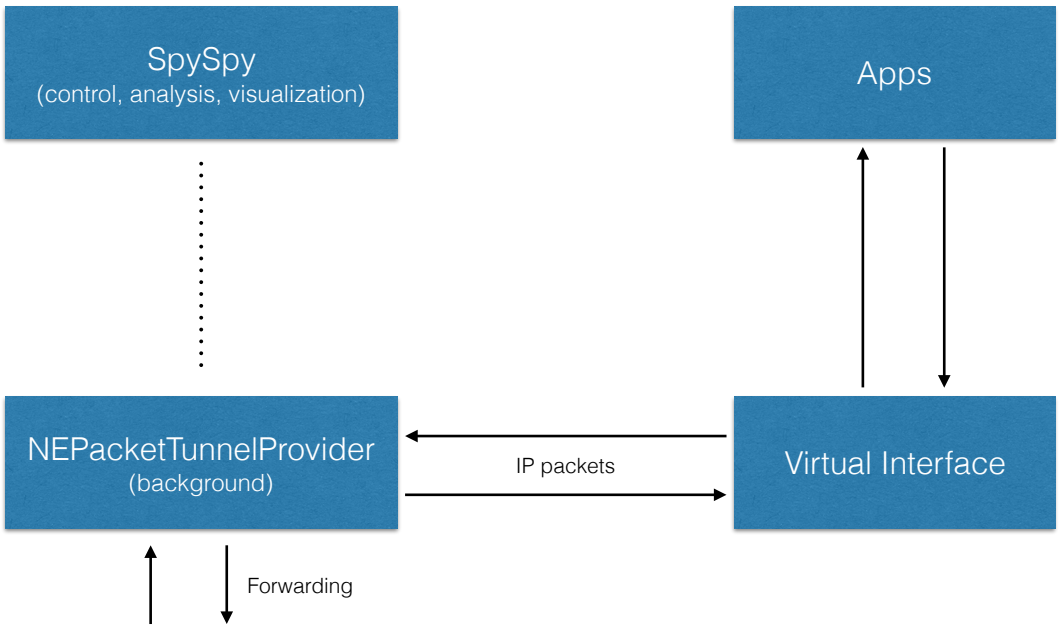

Figure 4.1: App design consisting of two parts: The SpySpy app (SpySpy box) that controls the monitoring. It is also responsible for the analysis and visualization. The other part is the NEPacketTunnelProvider (Apps, Virtual Interface and NEPacketTunnelProvider box) representing the traffic forwarding.

The app consists of two main parts (see Figure 4.1). Section 4.1.1 describes part one, the Spyspy app (SpySpy box), which interacts with the user and is responsible for the monitoring control, the analysis and the visualization. The second part (see Section 4.1.2) uses the NENetworkExtension framework that allows us to access and forward the traffic of all apps installed on the iPhone. The second part is represented by the three boxes NEPacketTunnelProvider, Virtual Interface and Apps in Figure 4.1. These parts communicate with each other using messages and shared storage symbolized by the dashed line between the SpySpy box and the NEPacketTunnelProvider box. The shared storage is realized by using App Groups (see Section 2.2). The app is developed for the new iOS 10 coming next fall. Therefore, we used the latest Xcode beta (Version 8.0 beta 2) and Swift 3. This section explains how the different classes interact with each other. Figure 4.2 shows a class diagram of all important classes used for the implementation. Dashed arrows mean that one class uses another one and solid arrows that a class is inherited from another one. Section 4.3 will present more detailed information about the visualization of the graphic user interface.

4.1.1 SpySpy App

Figure 4.2 shows all important classes needed for the implementation of the SpySpy app and the NEPacketTunnelProvider. Starting with the SpySpy app, the AppDelegate class is used as the interface between the app and the operating system. It provides functions to handle events such as executing some code during the app start or before the app will go to the background. It is also useful to provide functions for all view controllers. A view controller is a class that is used for each view provided in the graphic user interface. We implemented five of them: Three for the three tabs and two showing the detail view of the list items in the Statistics tab and the Domains tab. The AppDelegate class makes sure the first tab controller is called and that appropriate view controllers are closed and opened depending on the user's interaction. As you can see, there are three tabs each implemented using a tab controller class, which is inherited from the view controller class: The PacketTunnelViewController, the StatisticsTableViewController and the DomainsTableViewController. The first tab represented by the PacketTunnelViewController, called Overview, lets the user start and stop the traffic monitoring, reload and also reset the statistics. It also shows general statistics about the captured traffic. This controller is called first after the app is successfully started. It is responsible to check for an existing VPN configuration and to load it or to create a new one and to save it to the VPN configuration. It uses the PacketTunnelProvider class to configure and start the NEPacketTunnelProvider extension. The StatisticsTableViewController class represents the second tab. It shows detailed statistics to the user such as the amount of the monitored traffic regarding standard and unknown services. This controller is based on a list controller, meaning that it shows a list and each item is an object of a table item class. In this case, it is the StatisticsTableDetailsViewController class. Hitting an item in the list, the appropriate object is called and it shows detailed information about this item. The third tab, called Domains and represented by the DomainsTableViewController, shows also a list. It is a list of detected domains during the monitoring divided in domains with known leaks and domains without known leaks. The items of this list are objects of the DomainsTableDetailsViewController class. This Domains tab takes the result from the App Screening Phase, processes it and presents it to the user.
Due to the operating system design including sandboxing, it is not possible to just share data between the SpySpy app and the NEPacketTunnelProvider extension (see Section 2.2). Therefore, we use a feature called App Groups that lets us share data using a shared storage. It allows us to store and save arbitrary class objects. In Figure 4.2 you can see the StatisticsTraffic and the StatisticsTrafficManager class in the blue Statistics box. StatisticsTraffic objects can be stored in the App Group container and the StatisticsTrafficManager class is used as a manager for the shared storage. The AppDelegate class uses both classes in order to be able to read and write to the App Group. This class also provides a function to trigger a reload of the statistics to all view controllers and makes sure that the statistics are stored permanently. It uses the UserDefault class to achieve that no data is lost even if the app is closed or if it crashes. As soon as a statistics reload is triggered, the AppDelegate uses a messaging service to signal the NEPacketTunnelProvider class to store the current statistics into the App Group. After that, it can read the current statistics from this shared storage, update its own statistics and write it to the permanent storage. Such a statistic reload is triggered either when the user hits the 'reload

statistics' button or when the SpySpy app enters the foreground.
Some of Apple's frameworks can be used by every app. Other frameworks need special permissions. This is done using entitlements (see Section 2.2). The Frameworks box in Figure 4.2 shows only third party frameworks we used from developers providing their code for everyone online in public repositories. In order to use Apple's frameworks, we had to add the entitlements to the SpySpy app. The following two lists show third party framework licenses and entitlements that we added to the SpySpy app to use Apple's frameworks.

Framework Licenses

- **Charts**: Licensed under the Apache License. [28]
- **SwiftyJSON**: The MIT License (MIT). [27]
- **GCDAsyncSocket and GCDAsyncUDPSocket**: Public Domain License. The CocoaAsyncSocket project is in the public domain. [30]

Entitlements

- **VPN**: Personal VPN entitlement that allows the app to access the VPN configuration, to edit existing VPN configurations, to create new VPN configurations and to start/stop the VPN.
- **App Groups**: This entitlement allows to share data between the SpySpy app and the NEPacketTunnelProvider.
- **NENetworkExtension**: This special entitlement allows developers to use the NENetworkExtension. Developers have to apply for it on Apple's website to obtain it (see Section 2.2).

In order to present the results from the App Screening Phase to the user, we stored the results from the user study data analysis in JSON files, which are added to the SpySpy app during the build process. There is a JSON file for each leak category containing the domains and the corresponding apps of potential leaks and also if the data is leaked encrypted or unencrypted. Right after the app is started, the AppDelegate class loads all these JSON files in order to be able to process and correlate this data leaks with the traffic statistics to show the user potential domains, which might leak personal data. We also load a JSON file containing all standard services, meaning well-known TCP/UDP and destination port pairs. The AppDelegate and also other classes use the SwiftyJSON framework to load JSON structured data from a file or to process the data [26]. Another framework, which is used by most of the classes from the SpySpy app box, is the Charts framework by Daniel Gindi [28]. This framework allows to build nice charts including visualizing the data and results clearly for the user. We use a pie chart representation to present the traffic statistics.
Figure 4.2 also shows that classes of the SpySpy app box use many classes of the Statistics box that have not been mentioned yet. The StatisticsColorTemplate class provides defined color templates for the pie charts or for the tab controller bar. The classes DomainsTableEntries and StatisticsTableEntries are used to pass data of the specific list items to the detail view controllers. The AppAnalysisEntries class stores data from the JSON files to pass the data from the AppDelegate class to the DomainsTableDetailsViewController class.

4.1.2 NEPacketTunnelProvider Extension

This section explains the NEPacketTunnelProvider extension. Firstly, we introduce how the extension works and explain the custom-built UDP and TCP stack in user space in detail. After that, we show a TCP packet forwarding example and a UDP packet forwarding example in order to make clear how the forwarding works. Because Swift is still a new programming language, there are not many libraries for network programming. Everything like IP packet parsing, building an IP packet or parsing a DNS response had to be implemented ourselves.

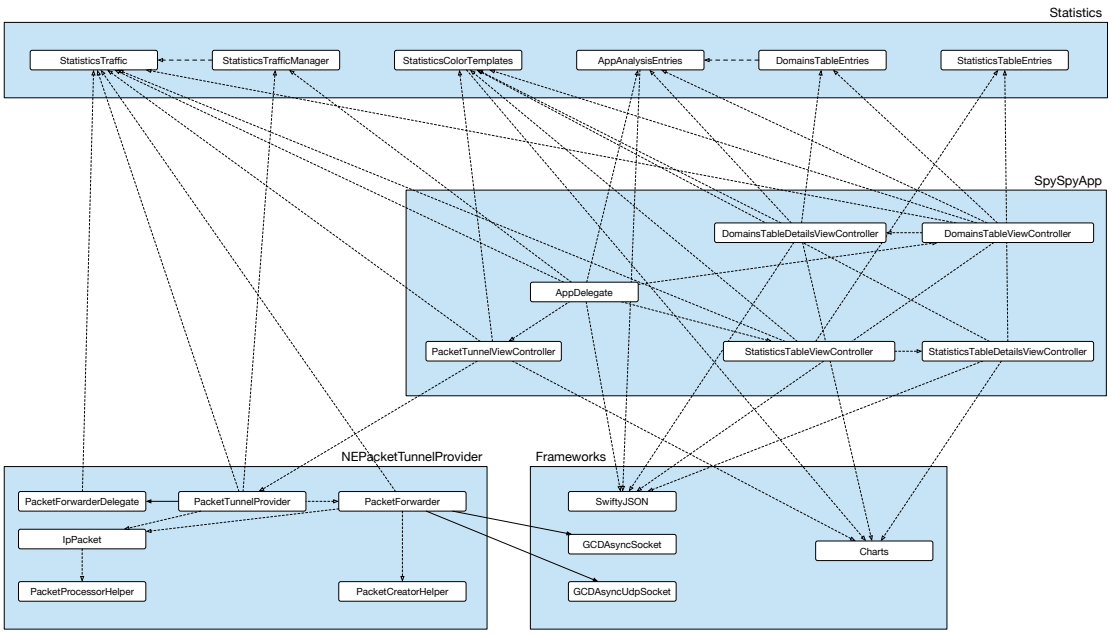

Figure 4.2: Class diagram of the important classes of the SpySpy app implementation. Dashed arrows mean that one class uses another one and solid arrows that a class is inherited from another one. The figure shows only classes written by us, by third party developers or inherited classes provided by Apple. It does not shown standard classes, which are automatically applied to the app when you create an app with Xcode. For example classes used to build the graphical user interface.

Overview

As mentioned before, NEPacketTunnelProvider is an extension that is configured and started by the SpySpy app and allows to reroute the traffic of every app to a virtual interface. The extension contains all the classes that you can see in the blue NEPacketTunnelProvider box in Figure 4.2. The PacketTunnelProvider class is the most important class provided by the Network Extension API to control the extension. It can configure, for example, the virtual interface with values such as a specific IP address and a DNS server. We configured the virtual interface to use Google's public DNS servers. These servers have IP addresses 8.8.8.8 and 8.8.4.4. As soon as the PacketTunnelProvider class creates the virtual interface, packets can be accessed using the NEPacketTunnelFlow class. This class allows to read and write packets to the virtual interface at the IP level. In order to work with packets, we created the IpPacket class. This class uses the PacketProcessorHelper to parse raw data packets and stores all important information as properties so that they can be easily accessed. The PacketCreatorHelper class, however, can be used to create new raw IP packets given a IpPacket object. The fact that the extension allows us to access packets at the IP layer is very important for the success of our SpySpy app. Nevertheless, it is also the biggest challenge we have to solve. The IP packets read from the NEPacketTunnelFlow have to be forwarded to the target server. This could be easily done by using a raw socket and by just writing the packets to this raw socket in order to let the operating system forward the packets as it is done by Haystack on Android for example [5]. Unfortunately, iOS does not allow anything that needs root permissions (see Section 2.2). Therefore, we were not able to use a raw socket and we have to forward the packets differently. It is possible to use standard TCP and UDP sockets within the NEPacketTunnelProvider extension, which allows us to send data to a target server. We used these sockets to forward UDP and TCP payload extracted from the IP packets to the target server. In order to handle packets coming from the virtual interface, we implemented a custom-built UDP and TCP stack in user space.

Custom-Built UDP and TCP Stack

The packet forwarding works as follows. For each detected packet of a new five tuple consisting of the source IP address, destination IP address, source port, destination port and protocol number (6 for UDP and 17 for TCP). The PacketTunnelProvider class creates a new PacketForwarder object of the PacketForwarder class. This PacketForwarder object includes a custom-built TCP stack to handle packets of the corresponding five tuple at the virtual interface side and a standard TCP socket to forward data to the target server with the corresponding destination IP and destination port. We use the GCDAsyncSocket class for TCP sockets and the GCDAsyncUdpSocket class for the UDP socket [29]. When the PacketForwarder object is created after receiving the first packet for a five tuple, we distinguish between creating a standard UDP socket for UDP packets or a standard TCP socket for TCP packets to establish a connection to the target server. The custom-built UDP or TCP stack handles packets at the virtual interface side. Handling UDP packets is not very complicated but maintaining a TCP connection with the applications is much more complibcated because TCP connections are stateful. This includes monitoring the TCP connection state such as sequence numbers, acknowledgment numbers or the TCP window size. The following code segment (Listing 4.1) shows the forwarding function of the PacketForwarder class. This function is the custom-built UDP and TCP stack. You can see that it processes all packets in the outgoingPacket array. This array contains all packets read from the virtual interface of the corresponding five tuple. For each packet in this array, it is checked if there is an existing standard socket. Handling a UDP packet means to check for an existing UDP socket, to use this socket to send only the UDP payload of the packet to the target server, to update the statistics and to remove the packet from the outgoingPacket array (see Listing 4.1, line 12). But as mentioned before, handling TCP packet is much more complicated. It starts just the same. Every packet of the outgointPacket array is processed by first checking if there is an existing standard TCP socket (see Listing 4.1, line 25). After that, we have to distinguish between TCP packets with different TCP flags. The following list presents the different cases depending on the flags of the received packet:

- **TCP SYN**: A TCP SYN packet is responded with a TCP SYN-ACK packet (see Listing 4.1, line 29). Therefore, we have to update the sequence and acknowledgment numbers, create a new IP packet using the PacketCreatorHelper class and add it to the incomingPackets array. All the packets in this array are written back to the virtual interface as soon as the writePacketsBack function is called.
- **TCP RST**: If a TCP RST packet is received (see Listing 4.1, line 38), we do nothing and wait for more packets.
- **TCP FIN**: A TCP FIN packet is handled as follows (see Listing 4.1, line 42). We have to update again the sequence and acknowledgment numbers, create a new TCP FIN-ACK packet but we can not write it back immediately. We have to wait two seconds until nothing is received at the standard TCP socket in order to make sure that we do not close the connection to early. Therefore, we use the DispatchTime class to set a timer when the TCP FIN-ACK packet should be sent. Each time we receive something at the standard TCP socket, two more seconds are added to this timer.
- **TCP ACK**: Receiving a TCP ACK packet containing data (see Listing 4.1, line 62) means that we update the sequence and acknowledgment numbers, create and write back a new TCP ACK packet to confirm the received packet. The payload is passed to the standard TCP socket.

After handling all these cases, we update the statistics for the TCP packet like the UDP packet and remove it from the outgoingPacket array. The PacketForwarder objects are not only responsible to forward the packet payload but also for updating the statistics. They update which destination port is used for which destination IP address and how much traffic is sent to or received from it. The objects also parse DNS responses for the requested domain and the IP addresses belonging to this domain. The approach to parse DNS responses is not fully tested and is based on some heuristics. Unfortunately, we did not have time to implement a more complicated approach.

```
\\
\\ This function, representing the custom-built UDP and TCP stack, is part of the PacketForwarder object and
\\ handles packets coming from the virtual interface. Distinguishing between UDP and TCP packets
\\ it uses a standard UDP or TCP socket to forward the packets to the target server.
\\
func forward() -> Void {

  while outgoingPackets.count > 0 {

    //
    // Custom-built UDP Stack: Handle UDP packets
    //
    if let socket = udpSocket {
      if let datagram = outgoingPackets[0].payload {
        socket.send(datagram, toHost: destinationAddress, port: UInt16(destinationPort)!, withTimeout: -1, tag: 555)
        // update domain statistics for outgoing traffic
        updateStatisticsOutgoing(packet: outgoingPackets[0].payload!)
        outgoingPackets.remove(at: 0)
      } else {
        NSLog("\(logName): ERROR: UDP: No payload for this IP packet") }
      }

    //
    // Custom-built TCP Stack: Handle TCP packets
    //
    } else if let socket = tcpSocket {
      if let datagram = outgoingPackets[0].payload {

        // handle a TCP SYN packet
        if Int(outgoingPackets[0].protoHeader["flagSyn"]!) > 0 {
          lastAckNumberLocal = lastSequNumberRemote + 1
          let newPacket = packetCreator.createIpPacket(Data(), newSourcePort: destinationPort, newSourceAddress: destinationAddress,
      newDestinationPort: sourcePort, newDestinationAddress: sourceAddress, newSequNumber: lastSequNumberLocal, newAckNumber:
      lastAckNumberLocal, newTcpFlags: "010010")
          incomingPacketsAddressFamily.append(NSNumber(value:Int(addressFamily)!))
          incomingPackets.append(newPacket)
          lastSequNumberLocal = lastSequNumberLocal + 1
          writePacketsBack()

        // handle a TCP RST packet -> nothing to do
        } else if Int(outgoingPackets[0].protoHeader["flagRst"]!) > 0 {
          continue

        // handle a TCP FIN packet
        } else if Int(outgoingPackets[0].protoHeader["flagAck"]!) > 0 &&
              Int(outgoingPackets[0].protoHeader["flagFin"]!) > 0 {
          let ackNumber = Int(self.outgoingPackets[0].protoHeader["sequenceNumber"]!)! + 1
          finAckPacket = self.packetCreator.createIpPacket(Data(), newSourcePort: self.destinationPort, newSourceAddress: self.
      destinationAddress, newDestinationPort: self.sourcePort, newDestinationAddress: self.sourceAddress, newSequNumber: self.
      lastSequNumberLocal, newAckNumber: ackNumber, newTcpFlags: "010001")
          // send TCP FIN after 2 seconds
          if dispatch == 0 {
            let deadlineTime = DispatchTime.now() + .seconds(2)
            DispatchQueue.main.after(when: deadlineTime) {
              self.dispatch -= 1
              if self.dispatch == 0 {
                self.incomingPacketsAddressFamily.append(NSNumber(value:Int(self.addressFamily)!))
                self.incomingPackets.append(self.finAckPacket!)
                self.writePacketsBack()
                self.timer = self.timer.addHours(hoursToAdd: -1)
              }
            }
            self.dispatch += 1
          }

        // handle a TCP ACK packet that includes data
        } else if Int(outgoingPackets[0].protoHeader["flagAck"]!) > 0 && datagram.count > 0 {
          lastAckNumberLocal = lastAckNumberLocal + datagram.count
          let newPacket = packetCreator.createIpPacket(Data(), newSourcePort: destinationPort, newSourceAddress: destinationAddress,
      newDestinationPort: sourcePort, newDestinationAddress: sourceAddress, newSequNumber: lastSequNumberLocal, newAckNumber:
      lastAckNumberLocal, newTcpFlags: "010000")
          incomingPacketsAddressFamily.append(NSNumber(value:Int(addressFamily)!))
          incomingPackets.append(newPacket)
          writePacketsBack()
          socket.write(outgoingPackets[0].payload, withTimeout: -1, tag: 555)
        }

        //
        // update domain statistics for outgoing traffic
        //
        updateStatisticsOutgoing(packet: outgoingPackets[0].payload!)
        outgoingPackets.remove(at: 0)

      } else {
        NSLog("\(logName): ERROR: TCP: No payload for this IP packet")
      }

    } else {
      NSLog("\(logName): ERROR: No socket")
    }

  }

}
```

Listing 4.1: This code segment shows the forwarding function of the PacketForwarder object representing the custom-built UDP and TCP stack in user space.

TCP Packets Forwarding Example

In order to make clear how the forwarding works in detail, we explain the process first with a TCP connection example. Figure 4.3 shows a flow chart of a TCP connection from a third party app to a target server and vice versa using our own custom-built TCP stack and a standard TCP socket. At the left side, one can see an app installed on the iPhone sending traffic. This can be any app, for example, '20min'. The box in the middle represents the NEPacketTunnelProvider

with the custom-built TCP stack and the standard TCP socket. The box to the right is the target server communicating with the standard TCP socket. Our custom-built TCP stack handles the packets being read from the virtual interface and the standard TCP socket is responsible to forward the payload received from the third party app to the target server. The custom-built TCP stack reads the IP packets from the virtual interface and parses the IP packets in order to get information from the IP and the TCP header. An object of the PacketForwarder class is created for each five tuple. For the example, we assume that there is an app sending traffic to a target server using one TCP connection and always the same source port. This means that one PacketForwarder is created. Using Figure 4.3, we want to explain how TCP packets are forwarded. The following list explains the forwarding each item representing an event.

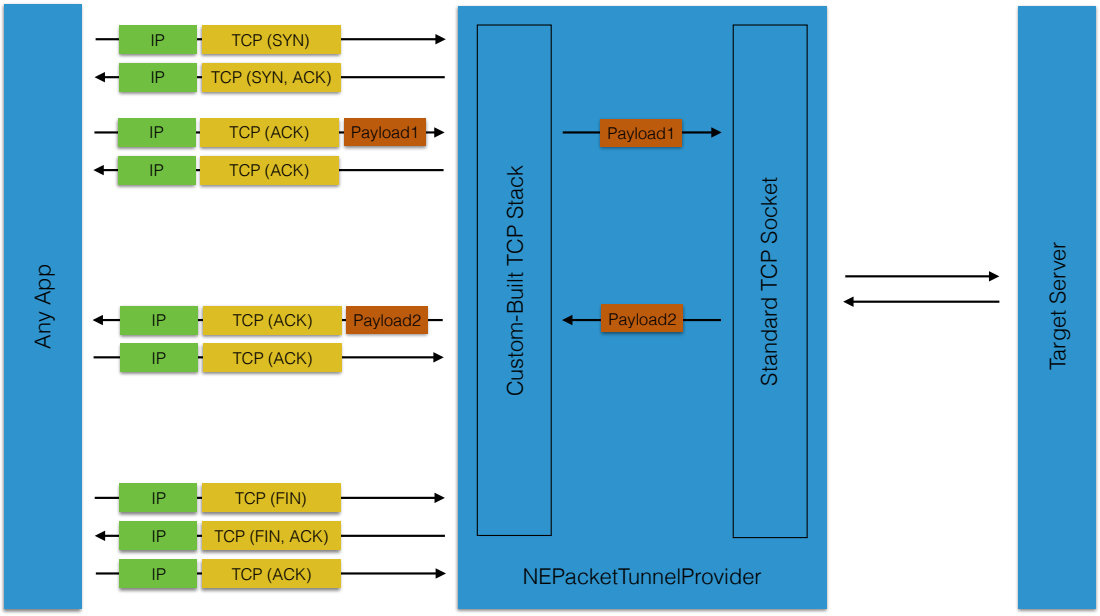

Figure 4.3: Custom-built TCP stack handling TCP packets coming from the virtual interface to forward them towards the target server and vice versa using a standard TCP socket.

- **The custom-built TCP stack receives a TCP SYN packet from the app**: If the app wants to send data to a target server over TCP, a TCP connection has to be established. Therefore, the app sends a TCP SYN packet. The custom-built TCP stack responds with a TCP SYN-ACK packet. The app responds with an ACK packet in order to finish the TCP handshake. Most of the time this final ACK packet of the handshake already contains data in the payload (see Payload1 in Figure 4.3). The custom-built TCP stack extracts the payload (Payload1) from the IP packet and only passes this payload (Payload1) to the standard TCP socket. The standard TCP socket sends the payload (Payload1) to the target server.
- **Standard TCP socket receives data from the target server**: As soon as the standard TCP socket receives some data from the target server, it passes the received payload (Payload2) back to the custom-built TCP stack, which creates a new IP packet with this payload (Payload2) and the corresponding IP and TCP header using the information stored in the PacketForwarder object. If the payload passed from standard TCP socket is to big to create a single IP packet, the custom-built TCP stack splits it into multiple payloads and creates an IP packet for each. After sending the IP packet with the payload (Payload2) or parts of it back to the app, the app responds with a TCP ACK packet. The custom-built TCP stack reads it and knows that the sent packet was received properly. This process is repeated for each TCP ACK packet sent from the app including some data until the app sends a TCP FIN packet.

- **The custom-built TCP stack receives a TCP FIN packet from the app**: Indicating that the app wants to close the TCP connection. After waiting until the standard TCP socket receives no more data from the target server for two seconds, the custom-built TCP stack responds with a TCP FIN-ACK packet. This means that both parties of the TCP connection want to close the TCP connection. Therefore, with receiving one last TCP ACK from the app, the TCP connection is closed, the standard TCP socket is closed and the PacketForwarder object is removed.

UDP Packets Forwarding Example

The forwarding of UDP packets works very similar to the TCP packet forwarding. A PacketForwarder object is created for any five tuple including a custom-built UDP stack and a standard UDP socket. Figure 4.4 shows how UDP packets are forwarded as follows. A UDP packet sent from any app is handled by the custom-built UDP stack, which extracts the payload (Payload1) and passes it to the standard UDP socket. This standard socket sends it to the target server. After the standard UDP socket receives payload from the target server, it passes the payload (Payload2) back to the custom-built UDP stack. A new IP packet is created and it is sent back to the particular app.

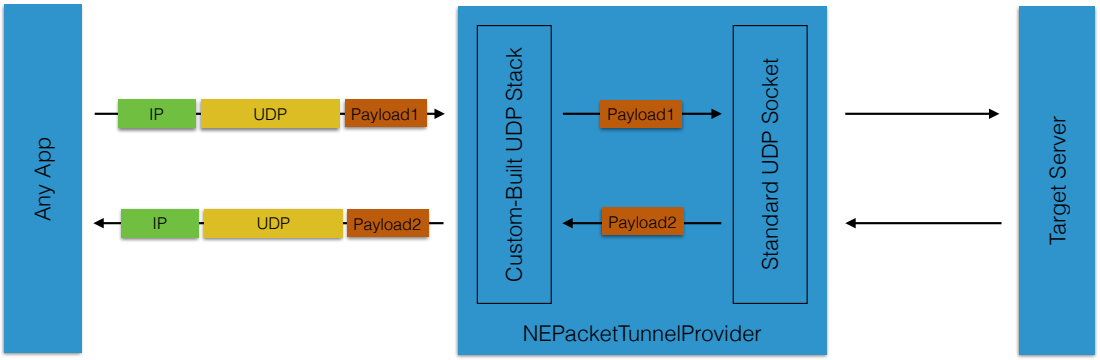

Figure 4.4: Custom-built UDP stack handling UDP packets coming from the virtual interface to forward them towards the target server and vice versa using a standard UDP socket

4.1.3 Design Restrictions

Our design brings along some drawbacks. Only UDP and TCP traffic can be forwarded. The extension can only be configured to route traffic based on the destination address to the virtual interface. Therefore, packets with other protocols like ICMP arrive at the virtual interface and can not be forwarded any further. Due to the assumption that most apps use services based either on the UDP or the TCP protocol, this only affects a very small part of the traffic. Another drawback is that SpySpy uses a VPN interface and therefore, no other VPN can be used at the same time. This might be a problem for company devices or people who want to protect their browsing using a VPN connection.

4.2 Testing

This section explains the process, which we used to verify that our custom-built UDP and TCP stacks forward the traffic properly. The standard UDP and TCP sockets and the target server are out-of-scope. Therefore, we focus in our tests on the traffic being read between the virtual interface and the NEPacketTunnelProvider. Unfortunately, it is not possible on iOS to capture traffic from a virtual interface and we had to implement a logging mechanism ourselves. Therefore, we logged every packet handled by the custom-built UDP or TCP stacks in a hexadecimal

representation to the console. This console can be accessed using Apple's IDE Xcode and running the SpySpy app on an iPhone connected to the same Macbook. Due to the console log limit, packets have to be divided in pieces smaller than 200 Bytes. We also logged a timestamp to each packet what allows us to analyze the timeline of logged packets. This console output can then be stored and processed. First of all, the script to process the output removes all log entries from the output except the once coming from the SpySpy app, which includes the once coming from the NEPacketTunnelProvider. After that, the splitted packets are reassembled and transformed to a special hexadecimal format defined by Wireshark. In order to include the timestamps, we have to process this hexadecimal format again to the pcap format. The resulting pcap file can be opened by the network protocol analyzer Wireshark. It lets you easily analyze the traffic on different network stack levels [31].
The following sections describe the tests we have performed. All tests are performed on an iPhone 5S running ios 10 beta 4 and using the Dolphin browser app [32] to request the desired websites. The Dolphin browser is used because it is easy to delete its cache. Due to caching on browsers like Safari or Google Chrome, it was not possible to access our test websites using HTTP after accessing it once using HTTPS. The SpySpy app has not been tested regarding IPv6 addresses. The testes performed in this thesis were all based on IPv4.

4.2.1 Test 1: Single HTTP Request

This test ensures that a single HTTP request caused by a website request can be forwarded. Table B.1 in Appendix B shows the packets recorded during this test. A HTTP request is used to get the content of a website containing only text. It allows us to control the UDP packet forwarding looking at the DNS responses. The HTTP request allows us also to make sure that TCP packets are treated properly. We can control sequence and acknowledgment numbers, TCP flags and more. The table shows some retransmissions of the TCP FIN-ACK packet coming from the Dolphin Browser. The reason is that the custom-built TCP stack is implemented to wait for two seconds before sending back a TCP FIN-ACK packet (see Section 4.1.2).

4.2.2 Test 2: Single HTTPS Request

This test is similar to the first test in Section 4.2.1. The only difference is that the website is loaded using SSL/TLS. A HTTPS request causes a SSL handshake. Table B.2 in Appendix B shows that the handshake is performed properly and that the website's content is sent to the Dolphin browser without any errors except the retransmission of Fin-Ack packets in the end, which we expect to happen because of the same reason we mentioned in Section 4.2.1.

4.2.3 Test 3: Parallel HTTP Requests

In order to make sure that not only loading simple websites works properly using SpySpy for monitoring, we also created a website with some text and four pictures. This should result in parallel HTTP requests. Table B.3 in Appendix B shows the traffic using Wireshark for the different HTTP requests. As you can see, every TCP connection is established and closed properly and all the content is downloaded properly. The table shows that there is a TCP connection established to send the initial HTTP request. After that the same connection is used to load one picture and three other TCP connections are established to load the other three pictures. In the end all four TCP connections are closed successfully.

4.2.4 Performance Test

Monitoring all the traffic requires the SpySpy app to forward all packets separately. This leads to a performance drop of the upload and download speed. In order to have an idea how much the monitoring slows down the iPhone's performance, we used a bandwidth testing app to measure upload and download speed. The iPhone is connected to a Wifi hotspot. The test is performed using an iPhone 5S running iOS 10 Developer Beta 4. Figure 4.5 presents the result of the bandwidth test without running the SpySpy app. The iPhone is able to download with a speed of around 33.68 Mbps and to upload with a speed of 16.06 Mbps. When the SpySpy app is running

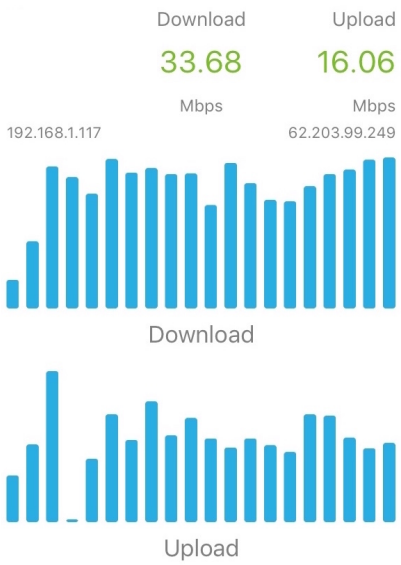

Figure 4.5: Bandwidth performance test performed by SpeedSpot [33]. One can see upload and download speed without running the SpySpy app and monitoring the traffic.

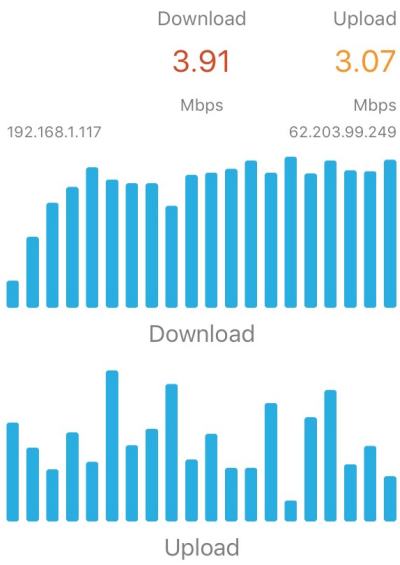

Figure 4.6: Bandwidth performance test performed by SpeedSpot [33]. One can see upload and download speed with running the SpySpy app and monitoring the traffic.

and monitoring all the traffic, the download speed drops to 3.91 Mbps and the upload speed to 3.07 Mbps. It looks like there is an upper limit of around 4 Mbps for the download speed and an upper limit of around 3 Mbps for the upload speed. Despite of the massive performance drop, the user does not notice a big loss of user performance. Maps are loaded a bit slower so that the user notices squares loaded separately. Video streaming, for example, with the Youtube app works very good without noticing anything. The following link leads to a demo video: `http://bit.ly/spyspydemovideo`.

4.3 SpySpy GUI

This section presents how we visualize the traffic statistics and results from the App Screening Phase regarding potential data leaks to the user and how we let the user control the monitoring. The SpySpy app is built based on a tab bar controller. This allows us to create different tabs with different content. There are three tabs: The Overview tab lets the user control the monitoring and shows general traffic information, the second tab called Statistics shows detailed statistics about the monitored traffic and the third tab presents the monitored domains and detailed information about personal data leaks to these domains.
Figure 4.7 shows the Overview tab. It provides an overview about the monitored traffic and also lets the user control the monitoring. There are four buttons at the bottom in the 'App Controller' part. Two of them let the user start and stop the monitoring and the other two reload or reset the traffic statistics. Reseting the traffic statistics deletes the entire permanently stored statistics. The upper part called 'Traffic Statistics' shows the date when the current monitoring session was started and the amount of traffic that was monitored during this session. The start date is only reset when the statistics are reset and not when the monitoring is stopped and started again. Screenshot 1 in Figure 4.7 shows the SpySpy app, which has not been started so far and has not monitored any traffic. That is why no data can be shown. After starting the monitoring and browsing some apps to monitor some traffic, the app will present the amount of traffic and also a pie chart of the monitored traffic distinguishing between encrypted and unencrypted traffic in percentage (see screenshot 2 in Figure 4.7). It is also written that we count the services with the following destination ports as encrypted: 443 for HTTPS, 989 and 990 for FTPS (FTP over TLS/SSL) and 22 for SSH. All other traffic from other services counts as unencrypted traffic.
The second tab showed in Figure 4.8 is called Statistics. It presents detailed traffic statis-

tics about the monitored traffic. So far, we added two statistics: One for standard services (screenshot 2 in Figure 4.8) and one for unknown services (screenshot 3 in Figure 4.8). Standard services are well-known UDP/TCP and port numbers combinations such as port 22 over TCP for the service SSH. The standard services are based on the '/etc/services' file from any Ubuntu operating system. This file combines well-known TCP/UDP and port combinations of sources like 'http://www.iana.org/assignments/port-numbers' and 'http://www.freebsd.org/cgi/cvsweb.cgi/src/etc/services'. All TCP/UDP and port combinations not appearing in this file are treated as unknown services. Both statistics show the amount of traffic in bytes using a pie chart representation and a short description bellow the pie chart.
The third tab is about showing a list of all monitored domains divided into two sections (see screenshot 1 in Figure 4.9). One section contains the domains with known leaks, where we know that some apps leak personal data to these domains. The other section contains the domains without known leaks. The domain list is created by looking at monitored DNS responses. Each of these two sections can be sorted either by bytes or by alphabet by clicking on the sort button in the upper bar on the right (screenshot 2). After clicking on a particular domain, details about this specific domain are revealed presented in screenshot 3. The pie chart at the top shows the amount of data sent to this domain in bytes divided in standard TCP/UDP services and others. Furthermore, the table in the lower part of the screenshot presents any personal data leak to this domain using the information we obtained during the App Screening Phase (see Chapter 3). The leaks are divided in the same categories used during the App Screening Phase and it also shows if the data is leaked unencrypted using HTTP or encrypted using HTTPS. The additional information button at the right to each category gives further details about potential apps that leak personal data of this particular category to this specific domain (see screenshot 4 in Figure 4.9). The last screenshot in Figure 4.9 is about IP addresses belonging to this domain, which we obtained from the DNS responses.

Figure 4.7: Screenshots of the Overview tab containing the monitoring control and general traffic statistics.

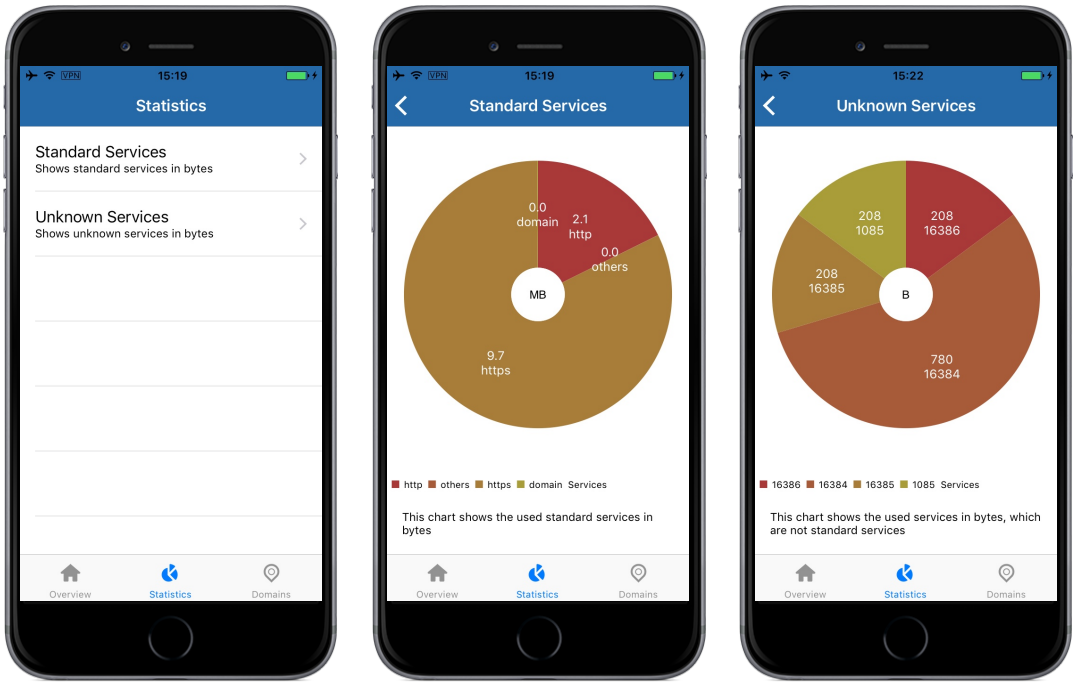

Figure 4.8: Screenshots of the Statistics tab containing detailed traffic statistics.

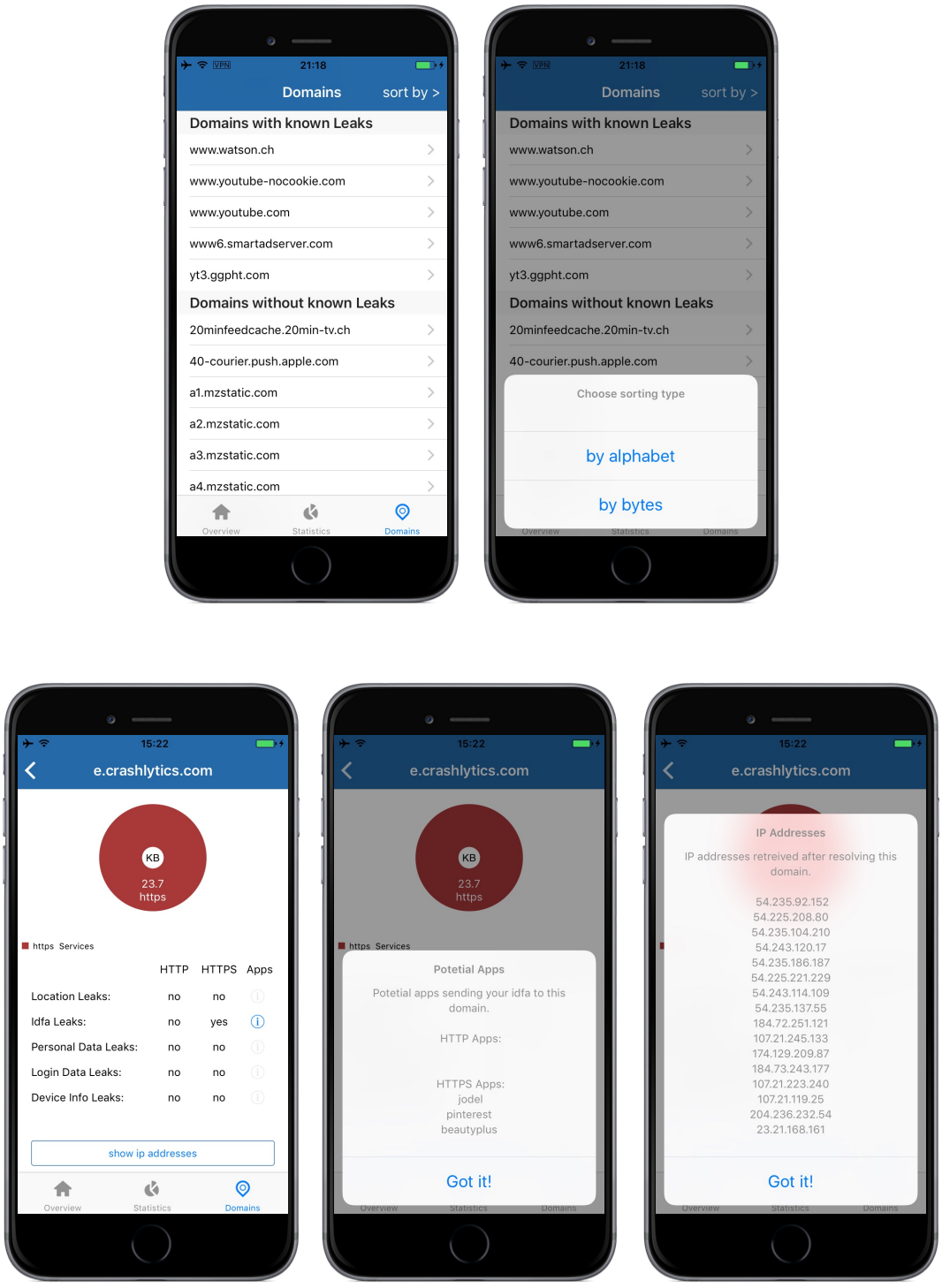

Figure 4.9: Screenshots of the Domains tab with a list of all monitored domains including details about potential apps that leak personal data to this particular domain.

Chapter 5

Summary and Outlook

We presented an approach for the iOS operating system to monitor all incoming and outgoing traffic directly on an iPhone and to identify apps leaking the user's personal data. In contrast to other approaches, SpySpy will be available for everyone because it does neither need a special firmware nor a jailbreaked iPhone. In order to keep the user safe, SpySpy does not intercept SSL/TLS connections and does not reroute the user's traffic to middle-boxes for the analysis. This makes identifying potential leakage of private data challenging as SpySpy cannot inspect the traffic. To this end, we present our approach consisting of the App Screening Phase and the Traffic Monitoring Phase.
Within the App Screening Phase, we created a special setup including a preconfigured iPhone and an Ubuntu server running a SSL-capable man-in-the-middle proxy server. Apps can be tested by test users using test user data, so that the test user's own personal data has not to be revealed. The captured traffic generated by the apps is analyzed regarding the leakage of personal data and categorized into different personal data leak categories. We performed two user studies using this setup to test iOS apps in a systematic way. User study 1 is about 60 very popular apps based on the Switzerland iTunes Charts and user study 2 is about different apps that in our opinion have a high potential to leak the user's location. The two user studies showed that more than half of the traffic is encrypted. Furthermore, almost all apps leak device information to third party domains and also almost half of the apps leak the user's location and the Identifier for Advertising to third party domains. Surprisingly, only two of all the apps leak personal data like contacts, calendar entries, reminders or images and also only to their own domain.
For the Monitoring Phase we implemented the SpySpy app running on an iPhone in order to monitor all incoming and outgoing traffic coming from any app installed on the iPhone. Apple's network extension NEPacketTunnelProvider allows us to access all the traffic using a virtual interface and to forward the traffic to the target servers. Due to operating system restrictions, we implemented a custom-built UDP and TCP stack to handle the traffic on the virtual interface side and used standard sockets to forward payload from and to the target servers. Finally, we create statistics about the monitored traffic and present the details about apps leaking personal data that we received from the App Screening Phase to the user. This helps users to decide which apps should be kept and which should be removed from the iPhone.
The App Screening Phase needs test users, which test the apps manually. They use preconfigured iPhones to test the apps. The generated traffic is captured and analyzed regarding leaks of personal data (see Chapter 3). In order to make the App Screening Phase scalable, we have to automatically get updates of new apps and test the apps automatically. A way to automate the testing on the phone and to avoid test users would be to use EggPlant Mobile [34]. EggPlant Mobile is a software provided by TestPlant that can be used to fully test mobile apps on iOS or Android using the iOS or Android Gateway agents to directly connect to live mobile devices or to emulators. TestPlant provides this software pointing out the unique image-based UI testing approach to test mobile applications and they also emphasize the ability to test multiple platforms with just one script and their software allows to fully control devices without any jailbreaking or modification of any app. Because the App Screening Phase has not been automated yet, the Traffic Monitoring Phase obtains only detailed information about potential apps leaking personal data for the apps tested in the two user studies. Therefore, the SpySpy app

only informs the user about domains and potential apps leaking personal data to these domains for a small number of apps.
The working SpySpy implementation allows us to monitor all incoming and outgoing traffic on the iPhone to show traffic statistics and potential apps leaking personal data to the user. Therefore, it is easy to add new approaches for detecting leaks. Furthermore, we only store traffic statistics on the user's iPhone so far. In order to aggregate statistics of different users, the SpySpy app has to be modified to let the user upload the statistics to a server in an anonymous way. In return we could provide always the current domain blacklist containing personal data leaks to the user as a motivation so that users upload their statistics. Finally, the SpySpy app creates a list of all seen domains and their appropriate IP addresses by parsing DNS responses. This would allow to compare it with a domain blacklist for example from Ad Block Plus and stop forwarding traffic to these blacklisted domains. These might help to prevent advertisements from being shown within apps.
All in all, the Traffic Monitoring Phase supported by the App Screening Phase can be used to monitor traffic of any app on an iPhone. The approach also points out apps tested during the App Screening Phase, which potentially leak the user's personal data and it also shows interesting general traffic statistics about the monitored traffic. The performed user studies also showed that only a few apps leak personal data like contacts or calendar entries to third party domains. However, personal data like the location, the Identifier for Advertising (IDFA) or specific device information are leaked from many apps to many own domains and even more third party domains.

Appendix A

App Screening Phase: User Study App Lists

A.1 User Study 1: App List based on iTunes Charts

No.	App Name	Details	Properly Working with SSL-Interception
1	20min	big news apps in ch	yes
2	akinator the genie	itunes charts switzerland (free top 50)	yes
3	angry birds action	itunes charts switzerland (free top 50)	yes
4	beautyplus	itunes charts switzerland (free top 50)	yes
5	booking.com	itunes charts switzerland (free top 50)	yes
6	brain it on - physics	itunes charts switzerland (free top 50)	yes
7	clash royale	itunes charts switzerland (free top 50)	no
8	color switch	itunes charts switzerland (free top 50)	yes
9	despicable me	http://www.techlicious.com/tip/the-worst-apps-for-privacy/	yes
10	dragon hills	itunes charts switzerland (free top 50)	yes
11	dream league soccer	itunes charts switzerland (free top 50)	yes
12	facebook	itunes charts switzerland (free top 50)	no
13	facebook messenger	itunes charts switzerland (free top 50)	no
14	flashlight	http://www.techlicious.com/tip/the-worst-apps-for-privacy/	yes
15	football star 2016	itunes charts switzerland (free top 50)	yes
16	free video - playlist	itunes charts switzerland (free top 50)	yes
17	frei musik player	itunes charts switzerland (free top 50)	yes
18	fruit ninja	http://www.techlicious.com/tip/the-worst-apps-for-privacy/	yes
19	gmail	itunes charts switzerland (free top 50)	yes
20	google - offizielle suchapp	itunes charts switzerland (free top 50)	yes
21	google maps	itunes charts switzerland (free top 50)	yes
22	google uebersetzer	itunes charts switzerland (free top 50)	yes
23	imusic es	itunes charts switzerland (free top 50)	yes
24	instagram	itunes charts switzerland (free top 50)	yes
25	itunes u	itunes charts switzerland (free top 50)	no
26	jodel	itunes charts switzerland (free top 50)	yes
27	kostenlos musik	itunes charts switzerland (free top 50)	yes
28	league of war	itunes charts switzerland (free top 50)	no
29	linkedin	popular social media app for business people	yes
30	lovoo	itunes charts switzerland (free)	yes
31	magic piano	itunes charts switzerland (free top 50)	yes
32	meteo swiss	itunes charts switzerland (free top 50)	yes
33	msqrd - live filter	itunes charts switzerland (free top 50)	yes
34	musical.ly	itunes charts switzerland (free top 50)	no
35	musik player	itunes charts switzerland (free top 50)	yes
36	piano tiles 2	itunes charts switzerland (free top 50)	yes
37	pinterest	itunes charts switzerland (free top 50)	yes
38	risky road	itunes charts switzerland (free top 50)	yes
39	sbb mobile	itunes charts switzerland (free top 50)	yes
40	sbb mobile preview	itunes charts switzerland (free top 50)	no
41	schweizer fleisch	itunes charts switzerland (free top 50)	yes
42	shazam	itunes charts switzerland (free top 50)	yes
43	skype	itunes charts switzerland (free top 50)	no
44	slither.io	itunes charts switzerland (free top 50)	no
45	slither.ioR	itunes charts switzerland (free top 50)	no

No.	App Name	Details	Properly Working with SSL-Interception
46	snapchat	itunes charts switzerland (free top 50)	no
47	spotify	itunes charts switzerland (free top 50)	no
48	stack	itunes charts switzerland (free top 50)	yes
49	tagesanzeiger	big news apps in ch	yes
50	tinder	itunes charts switzerland (free)	yes
51	tripadvisor hotels	itunes charts switzerland (free top 50)	yes
52	uber	itunes charts switzerland (free top 50)	yes
53	viber	itunes charts switzerland (free top 50)	yes
54	visage lab	itunes charts switzerland (free top 50)	yes
55	watson	big news apps in ch	yes
56	whatsapp	itunes charts switzerland (free top 50)	no
57	wish	itunes charts switzerland (free top 50)	yes
58	youtube	itunes charts switzerland (free top 50)	yes
59	zalando	itunes charts switzerland (free top 50)	yes
60	zattoo live tv	itunes charts switzerland (free)	yes
	Total of not properly working apps		13

Table A.1: App list based on top 50 of the Switzerland iTunes charts for free apps plus 10 apps with a high potential to leak personal data. The table also shows which app worked with our test setup using a SSL-capable proxy and which do not work. In total 47 apps work properly and 13 do not due to certificate pinning.

A.2 User Study 2: App List of Apps potentially leaking the Location

No.	App Name	Request Location Permission	Properly Working with SSL-Interception
1	agent-x	yes	yes
2	airbnb	yes	yes
3	amaze	yes	yes
4	autoindex	no	yes
5	cheap flights here	yes	no
6	citymapper	yes	no
7	citymapper	yes	yes
8	cyclemeter	yes	yes
9	dropbox	no	no
10	easyjet.mobile	yes	no
11	ebookers	yes	yes
12	feedly	yes	yes
13	flights	yes	yes
14	forecast	yes	yes
15	foursquare	yes	yes
16	free gps	yes	yes
17	galileo	yes	yes
18	gc tools	yes	yes
19	geocaching	yes	yes
20	glympse	no	no
21	gps logbooks	yes	yes
22	gps-buddy	?	no
23	gps-r	yes	yes
24	here maps	yes	yes
25	home.ch	yes	yes
26	homegate.ch	yes	yes
27	imate chinese	yes	yes
28	ingress	yes	yes
29	inrix traffic	yes	yes
30	inroute	yes	yes
31	jetradar	yes	yes
32	landi wetter	yes	yes
33	landlord2	yes	yes
34	layar	no	yes
35	magellan active	?	no
36	meteoblue	yes	yes
37	mivue	?	no
38	mobile network sunrise	yes	yes
39	myswisscom	no	yes
40	mytracks	yes	yes
41	myupc	no	no
42	navmii switzerland	yes	yes
43	nike running	yes	yes
44	on the fly	yes	yes
45	orbitz	yes	yes
46	planet 105	yes	yes
47	parallel kingdom	?	no
48	parallel mafia	?	no
49	priceline.com	yes	yes
50	property comparis	yes	yes
51	raincoat	yes	yes
52	ricardo.ch	yes	yes
53	runkeeper	yes	yes
54	runmeter gps	yes	yes
55	runners free	yes	yes
56	runtastic	yes	yes
57	scout	yes	yes
58	scout lite	yes	yes
59	secret escape	yes	yes
60	sickweather	yes	yes
61	silicon.de	yes	yes
62	simple gps	yes	yes
63	slickdeals	no	yes
64	snapguide	yes	yes
65	speedometer	yes	yes
66	speedsmart wifi	yes	yes

No.	App Name	Request Location Permission	Properly Working with SSL-Interception
67	speedtest	yes	yes
68	speedtest.pro	yes	yes
69	srf meteo	yes	yes
70	stepz	no	yes
71	strava	yes	yes
72	stucard	yes	yes
73	swarm	yes	yes
74	swiss snow	yes	yes
75	sygic	?	no
76	toll lookout gps	yes	yes
77	tomtom go	?	no
78	topo maps	yes	yes
79	tourality	?	no
80	tracker.com	yes	yes
81	tracks logger	yes	yes
82	trails	yes	yes
83	tripit	yes	yes
84	triposo	yes	yes
85	tuneln	yes	no
86	turf.ly	yes	yes
87	turfwars	yes	yes
88	uepaa	yes	yes
89	untapped	yes	yes
90	vivino	?	no
91	wanderbock	yes	no
92	waze	yes	yes
93	where am i	yes	yes
94	where am i at	yes	yes
95	white risk	no	yes
96	yahoo weather	yes	yes
97	zaploot	yes	yes
98	zombies run	yes	yes
	Total of not properly working apps		17

Table A.2: App list of the apps with a high potential to leak the location. The table also shows which app worked with our test setup using a SSL-capable proxy and which do not work. In total 81 apps work properly and 17 do not due to certificate pinning. It also shows which apps ask the user for the permission to access the location.

Appendix B

Traffic Monitoring Phase: SpySpy App Testing

B.1 Test 1: Single HTTP Request

Time	Source	Destination	Proto	Src Port	Dest Port	Length	Info
8.491000	10.0.0.20	8.8.8.8	DNS	53650	53	69	Standard query 0x6b68 A n.ethz.ch
8.571000	8.8.8.8	10.0.0.20	DNS	domain	53650	116	Standard query response 0x6b68 CNAME idwebhost 202 24.ethz.ch A 129.132.202.24
8.579000	10.0.0.20	129.132.202.24	TCP	61783	80	78	61783 > http [SYN ECN CWR] Seq=1674369100 Win=65535 Len=0 MSS=1460 WS=32 TSval=1157404365 TSecr=0 SACK_PERM=1
8.581000	129.132.202.24	10.0.0.20	TCP	http	61783	54	http > 61783 [SYN ACK] Seq=15322 Ack=1674369101 Win=65535 Len=0
8.583000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [ACK] Seq=1674369101 Ack=15323 Win=2097120 Len=0
8.589000	10.0.0.20	129.132.202.24	HTTP	61783	80	447	GET / amreinse/spyspy/test2_http HTTP/1.1
8.592000	129.132.202.24	10.0.0.20	TCP	http	61783	54	http > 61783 [ACK] Seq=15323 Ack=1674369494 Win=65535 Len=0
8.669000	129.132.202.24	10.0.0.20	TCP	http	61783	351	[TCP segment of a reassembled PDU]
8.676000	129.132.202.24	10.0.0.20	TCP	http	61783	1502	[TCP segment of a reassembled PDU]
8.684000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [ACK] Seq=1674369494 Ack=17068 Win=2097120 Len=0
8.699000	129.132.202.24	10.0.0.20	TCP	http	61783	5054	[TCP segment of a reassembled PDU]
8.734000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [ACK] Seq=1674369494 Ack=22068 Win=2097120 Len=0
8.742000	129.132.202.24	10.0.0.20	TCP	http	61783	846	[TCP segment of a reassembled PDU]
8.756000	129.132.202.24	10.0.0.20	TCP	http	61783	5054	[TCP segment of a reassembled PDU]
8.779000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [ACK] Seq=1674369494 Ack=27860 Win=2097120 Len=0
8.788000	129.132.202.24	10.0.0.20	TCP	http	61783	3246	[TCP segment of a reassembled PDU]
8.806000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [ACK] Seq=1674369494 Ack=31052 Win=2097120 Len=0
8.818000	129.132.202.24	10.0.0.20	TCP	http	61783	5054	[TCP segment of a reassembled PDU]
8.844000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [ACK] Seq=1674369494 Ack=36052 Win=2097120 Len=0
8.853000	129.132.202.24	10.0.0.20	TCP	http	61783	5054	[TCP segment of a reassembled PDU]
8.879000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [ACK] Seq=1674369494 Ack=41052 Win=2097120 Len=0
8.890000	129.132.202.24	10.0.0.20	TCP	http	61783	5054	[TCP segment of a reassembled PDU]
8.917000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [ACK] Seq=1674369494 Ack=46052 Win=2097120 Len=0
8.928000	129.132.202.24	10.0.0.20	TCP	http	61783	5054	[TCP segment of a reassembled PDU]
8.955000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [ACK] Seq=1674369494 Ack=51052 Win=2097120 Len=0
8.965000	129.132.202.24	10.0.0.20	TCP	http	61783	5054	[TCP segment of a reassembled PDU]
9.008000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [ACK] Seq=1674369494 Ack=56052 Win=2097120 Len=0
9.016000	129.132.202.24	10.0.0.20	TCP	http	61783	5054	[TCP segment of a reassembled PDU]
9.060000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [ACK] Seq=1674369494 Ack=61052 Win=2097120 Len=0
9.071000	129.132.202.24	10.0.0.20	TCP	http	61783	5054	[TCP segment of a reassembled PDU]
9.104000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [ACK] Seq=1674369494 Ack=66052 Win=2097120 Len=0
9.116000	129.132.202.24	10.0.0.20	TCP	http	61783	5054	[TCP segment of a reassembled PDU]
9.150000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [ACK] Seq=1674369494 Ack=71052 Win=2097120 Len=0
9.163000	129.132.202.24	10.0.0.20	TCP	http	61783	5054	[TCP segment of a reassembled PDU]
9.198000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [ACK] Seq=1674369494 Ack=76052 Win=2097120 Len=0
9.212000	129.132.202.24	10.0.0.20	HTTP	http	61783	2902	HTTP/1.1 200 OK (text/html)
9.227000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [ACK] Seq=1674369494 Ack=78900 Win=2097120 Len=0
15.369000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [FIN ACK] Seq=1674369494 Ack=78900 Win=2097120 Len=0
15.678000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [FIN ACK] Seq=1674369494 Ack=78900 Win=2097120 Len=0
16.088000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [FIN ACK] Seq=1674369494 Ack=78900 Win=2097120 Len=0
16.698000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [FIN ACK] Seq=1674369494 Ack=78900 Win=2097120 Len=0
17.537000	129.132.202.24	10.0.0.20	TCP	http	61783	54	http > 61783 [FIN ACK] Seq=78900 Ack=1674369495 Win=65535 Len=0
17.563000	10.0.0.20	129.132.202.24	TCP	61783	80	54	61783 > http [ACK] Seq=1674369495 Ack=78901 Win=2097120 Len=0

Table B.1: This Table shows the captured traffic of Test 1. This test shows that it is possible to use a single HTTP request to get the content of a website containing only text. The table shows three IP addresses. IP Address 10.0.0.20 is the IP address used by the virtual interface, 8.8.8.8 is the virtual interface's configured DNS server and 129.132.202.24 is the IP address of the target server. There are also TCP packets with a length of 5054 bytes. These are parts of a big TCP segment, which was split by the custom-built TCP stack. The note 'TCP segment of a reassembled PDU' means that the different parts of the TCP segment have to be reassembled in order to obtain the entire TCP segment.

B.2 Test 2: Single HTTPS Request

Time	Source	Destination	Proto	Src Port	Dest Port	Length	Info
9.394000	10.0.0.20	129.132.202.24	TCP	63664	443	78	63664 > https [SYN] Seq=852583389 Win=65535 Len=0 MSS=1460 WS=32 TSval=1167017714 TSecr=0 SACK_PERM=1
9.396000	129.132.202.24	10.0.0.20	TCP	https	63664	54	https > 63664 [SYN ACK] Seq=28595 Ack=852583390 Win=65535 Len=0
9.399000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [ACK] Seq=852583390 Ack=28596 Win=2097120 Len=0
9.406000	10.0.0.20	129.132.202.24	TLSv1.2	63664	443	258	Client Hello
9.408000	129.132.202.24	10.0.0.20	TCP	https	63664	54	https > 63664 [ACK] Seq=28596 Ack=852583594 Win=65535 Len=0
9.431000	129.132.202.24	10.0.0.20	TLSv1.2	https	63664	1502	Server Hello
9.443000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [ACK] Seq=852583594 Ack=30044 Win=2097120 Len=0
9.448000	129.132.202.24	10.0.0.20	TLSv1.2	https	63664	1502	Certificate
9.458000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [ACK] Seq=852583594 Ack=31492 Win=2097120 Len=0
9.462000	129.132.202.24	10.0.0.20	TLSv1.2	https	63664	249	Server Key Exchange
9.477000	10.0.0.20	129.132.202.24	TLSv1.2	63664	443	129	Client Key Exchange
9.478000	129.132.202.24	10.0.0.20	TCP	https	63664	54	https > 63664 [ACK] Seq=31687 Ack=852583669 Win=65535 Len=0
9.480000	10.0.0.20	129.132.202.24	TLSv1.2	63664	443	60	Change Cipher Spec
9.481000	129.132.202.24	10.0.0.20	TCP	https	63664	54	https > 63664 [ACK] Seq=31687 Ack=852583675 Win=65535 Len=0
9.482000	10.0.0.20	129.132.202.24	TLSv1.2	63664	443	99	Encrypted Handshake Message
9.483000	129.132.202.24	10.0.0.20	TCP	https	63664	54	https > 63664 [ACK] Seq=31687 Ack=852583720 Win=65535 Len=0
9.540000	129.132.202.24	10.0.0.20	TLSv1.2	https	63664	105	Change Cipher Spec Encrypted Handshake Message
9.545000	10.0.0.20	129.132.202.24	TLSv1.2	63664	443	447	Application Data
9.547000	129.132.202.24	10.0.0.20	TCP	https	63664	54	https > 63664 [ACK] Seq=31738 Ack=852584113 Win=65535 Len=0
9.638000	129.132.202.24	10.0.0.20	TLSv1.2	https	63664	1502	Application Data
9.648000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [ACK] Seq=852584113 Ack=33186 Win=2097120 Len=0
9.659000	129.132.202.24	10.0.0.20	TCP	https	63664	1502	[TCP segment of a reassembled PDU]
9.668000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [ACK] Seq=852584113 Ack=34634 Win=2097120 Len=0
9.682000	129.132.202.24	10.0.0.20	TCP	https	63664	5054	[TCP segment of a reassembled PDU]
9.712000	129.132.202.24	10.0.0.20	TLSv1.2	https	63664	5054	Application Data
9.735000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [ACK] Seq=852584113 Ack=44634 Win=2097120 Len=0
9.749000	129.132.202.24	10.0.0.20	TLSv1.2	https	63664	5054	Application Data
9.785000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [ACK] Seq=852584113 Ack=49634 Win=2097120 Len=0
9.792000	129.132.202.24	10.0.0.20	TCP	https	63664	722	[TCP segment of a reassembled PDU]
9.818000	129.132.202.24	10.0.0.20	TCP	https	63664	5054	[TCP segment of a reassembled PDU]
9.852000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [ACK] Seq=852584113 Ack=55302 Win=2097120 Len=0
9.862000	129.132.202.24	10.0.0.20	TLSv1.2	https	63664	5054	Application Data
9.883000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [ACK] Seq=852584113 Ack=60302 Win=2097120 Len=0
9.893000	129.132.202.24	10.0.0.20	TLSv1.2	https	63664	5054	Application Data
9.916000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [ACK] Seq=852584113 Ack=65302 Win=2097120 Len=0
9.927000	129.132.202.24	10.0.0.20	TCP	https	63664	5054	[TCP segment of a reassembled PDU]
9.951000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [ACK] Seq=852584113 Ack=70302 Win=2097120 Len=0
9.960000	129.132.202.24	10.0.0.20	TLSv1.2	https	63664	5054	Application Data
9.985000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [ACK] Seq=852584113 Ack=75302 Win=2097120 Len=0
9.995000	129.132.202.24	10.0.0.20	TCP	https	63664	5054	[TCP segment of a reassembled PDU]
10.019000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [ACK] Seq=852584113 Ack=80302 Win=2097120 Len=0
10.030000	129.132.202.24	10.0.0.20	TLSv1.2	https	63664	5054	Application Data
10.069000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [ACK] Seq=852584113 Ack=85302 Win=2097120 Len=0
10.078000	129.132.202.24	10.0.0.20	TCP	https	63664	1254	[TCP segment of a reassembled PDU]
10.088000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [ACK] Seq=852584113 Ack=86502 Win=2097120 Len=0
10.113000	129.132.202.24	10.0.0.20	TLSv1.2	https	63664	5054	Application Data
10.157000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [ACK] Seq=852584113 Ack=91502 Win=2097120 Len=0
10.170000	129.132.202.24	10.0.0.20	TLSv1.2	https	63664	4202	Application Data
10.203000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [ACK] Seq=852584113 Ack=95650 Win=2097120 Len=0
17.500000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [FIN ACK] Seq=852584113 Ack=95650 Win=2097120 Len=0
17.808000	10.0.0.20	129.132.202.24	TCP	63664	443	54	[TCP Retransmission] 63664 > https [FIN ACK] Seq=852584113 Ack=95650 Win=2097120 Len=0
18.214000	10.0.0.20	129.132.202.24	TCP	63664	443	54	[TCP Retransmission] 63664 > https [FIN ACK] Seq=852584113 Ack=95650 Win=2097120 Len=0
18.818000	10.0.0.20	129.132.202.24	TCP	63664	443	54	[TCP Retransmission] 63664 > https [FIN ACK] Seq=852584113 Ack=95650 Win=2097120 Len=0
19.705000	129.132.202.24	10.0.0.20	TCP	https	63664	54	https > 63664 [FIN ACK] Seq=95650 Ack=852584114 Win=65535 Len=0
19.732000	10.0.0.20	129.132.202.24	TCP	63664	443	54	63664 > https [ACK] Seq=852584114 Ack=95651 Win=2097120 Len=0

Table B.2: This table shows the captured traffic of Test 2. This test shows that it is possible to use a single HTTP request to get the content of a website containing only text. The table shows two IP addresses. IP Address 10.0.0.20 is the IP address used by the virtual interface and 129.132.202.24 is the IP address of the target server. There are also TCP packets with a length of 5054 bytes. These are parts of a big TCP segment, which was split by the custom-built TCP stack. The note 'TCP segment of a reassembled PDU' means that the different parts of the TCP segment have to be reassembled in order to obtain the entire TCP segment.

B.3 Test 3: Parallel HTTP Requests

Time	Source	Destination	Proto	Src Port	Dest Port	Length	Info
0.250000	10.0.0.20	129.132.202.24	TCP	63873	80	78	63873 > http [SYN] Seq=2746999895 Win=65535 Len=0 MSS=1460 WS=32 TSval=1168063181 TSecr=0 SACK_PERM=1
0.252000	129.132.202.24	10.0.0.20	TCP	http	63873	54	http > 63873 [SYN ACK] Seq=4493 Ack=2746999896 Win=65535 Len=0
0.255000	10.0.0.20	129.132.202.24	TCP	63873	80	54	63873 > http [ACK] Seq=2746999896 Ack=4494 Win=2097120 Len=0
0.259000	10.0.0.20	129.132.202.24	HTTP	63873	80	447	GET / amreinse/spyspy/test4_http HTTP/1.1
0.263000	129.132.202.24	10.0.0.20	TCP	http	63873	54	http > 63873 [ACK] Seq=4494 Ack=2747000289 Win=65535 Len=0
0.314000	129.132.202.24	10.0.0.20	TCP	http	63873	348	[TCP segment of a reassembled PDU]
0.318000	129.132.202.24	10.0.0.20	HTTP	http	63873	556	HTTP/1.1 200 OK (text/html)
0.321000	10.0.0.20	129.132.202.24	TCP	63873	80	54	63873 > http [ACK] Seq=2747000289 Ack=5290 Win=2097120 Len=0
0.353000	10.0.0.20	129.132.202.24	HTTP	63873	80	424	GET / amreinse/spyspy/pictures/cat929292.jpg HTTP/1.1
0.354000	129.132.202.24	10.0.0.20	TCP	http	63873	54	http > 63873 [ACK] Seq=5290 Ack=2747000659 Win=65535 Len=0
0.367000	10.0.0.20	129.132.202.24	TCP	63875	80	78	63875 > http [SYN] Seq=4023138972 Win=65535 Len=0 MSS=1460 WS=32 TSval=1168063286 TSecr=0 SACK_PERM=1
0.369000	129.132.202.24	10.0.0.20	TCP	http	63875	54	http > 63875 [SYN ACK] Seq=57633 Ack=4023138973 Win=65535 Len=0
0.372000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [ACK] Seq=4023138973 Ack=57634 Win=2097120 Len=0
0.378000	10.0.0.20	129.132.202.24	HTTP	63875	80	425	GET / amreinse/spyspy/pictures/dog9283882.jpg HTTP/1.1
0.382000	129.132.202.24	10.0.0.20	TCP	http	63875	54	http > 63875 [ACK] Seq=57634 Ack=4023139344 Win=65535 Len=0
0.391000	129.132.202.24	10.0.0.20	TCP	http	63873	4694	[TCP segment of a reassembled PDU]
0.416000	10.0.0.20	129.132.202.24	TCP	63877	80	78	63877 > http [SYN] Seq=292528685 Win=65535 Len=0 MSS=1460 WS=32 TSval=1168063326 TSecr=0 SACK_PERM=1
0.418000	129.132.202.24	10.0.0.20	TCP	http	63877	54	http > 63877 [SYN ACK] Seq=1627 Ack=292528686 Win=65535 Len=0
0.421000	10.0.0.20	129.132.202.24	TCP	63877	80	54	63877 > http [ACK] Seq=292528686 Ack=1628 Win=2097120 Len=0
0.426000	10.0.0.20	129.132.202.24	HTTP	63877	80	425	GET / amreinse/spyspy/pictures/fox9392929.jpg HTTP/1.1
0.432000	129.132.202.24	10.0.0.20	TCP	http	63877	54	http > 63877 [ACK] Seq=1628 Ack=292529057 Win=65535 Len=0
0.443000	10.0.0.20	129.132.202.24	TCP	63879	80	78	63879 > http [SYN ECN CWR] Seq=947935509 Win=65535 Len=0 MSS=1460 WS=32 TSval=1168063349 TSecr=0 SACK_PERM=1
0.448000	129.132.202.24	10.0.0.20	TCP	http	63879	54	http > 63879 [SYN ACK] Seq=32506 Ack=947935510 Win=65535 Len=0
0.451000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935510 Ack=32507 Win=2097120 Len=0
0.458000	10.0.0.20	129.132.202.24	HTTP	63879	80	426	GET / amreinse/spyspy/pictures/lion2938281.jpg HTTP/1.1
0.460000	129.132.202.24	10.0.0.20	TCP	http	63879	54	http > 63879 [ACK] Seq=32507 Ack=947935882 Win=65535 Len=0
0.502000	10.0.0.20	129.132.202.24	TCP	63873	80	54	63873 > http [ACK] Seq=2747000659 Ack=9930 Win=2097120 Len=0
0.520000	129.132.202.24	10.0.0.20	TCP	http	63875	353	[TCP segment of a reassembled PDU]
0.528000	129.132.202.24	10.0.0.20	TCP	http	63877	1800	[TCP segment of a reassembled PDU]
0.539000	10.0.0.20	129.132.202.24	TCP	63877	80	54	63877 > http [ACK] Seq=292529057 Ack=3374 Win=2097120 Len=0
0.552000	129.132.202.24	10.0.0.20	TCP	http	63879	5054	[TCP segment of a reassembled PDU]
0.575000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935882 Ack=37507 Win=2097120 Len=0
0.588000	129.132.202.24	10.0.0.20	TCP	http	63879	5054	[TCP segment of a reassembled PDU]
0.626000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935882 Ack=42507 Win=2097120 Len=0
0.640000	129.132.202.24	10.0.0.20	TCP	http	63879	2889	[TCP segment of a reassembled PDU]
0.665000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935882 Ack=45342 Win=2097120 Len=0
0.668000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [ACK] Seq=4023139344 Ack=57933 Win=2097120 Len=0
0.680000	129.132.202.24	10.0.0.20	HTTP	http	63873	3528	HTTP/1.1 200 OK (JPEG JFIF image)
0.703000	10.0.0.20	129.132.202.24	TCP	63873	80	54	63873 > http [ACK] Seq=2747000659 Ack=13404 Win=2097120 Len=0
0.718000	129.132.202.24	10.0.0.20	TCP	http	63875	5054	[TCP segment of a reassembled PDU]
0.756000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [ACK] Seq=4023139344 Ack=62933 Win=2097120 Len=0
0.771000	129.132.202.24	10.0.0.20	TCP	http	63875	5054	[TCP segment of a reassembled PDU]
0.806000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [ACK] Seq=4023139344 Ack=67933 Win=2097120 Len=0
0.817000	129.132.202.24	10.0.0.20	TCP	http	63875	5054	[TCP segment of a reassembled PDU]
0.842000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [ACK] Seq=4023139344 Ack=72933 Win=2097120 Len=0
0.853000	129.132.202.24	10.0.0.20	TCP	http	63875	5054	[TCP segment of a reassembled PDU]
0.880000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [ACK] Seq=4023139344 Ack=77933 Win=2097120 Len=0
0.890000	129.132.202.24	10.0.0.20	TCP	http	63875	5054	[TCP segment of a reassembled PDU]
0.918000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [ACK] Seq=4023139344 Ack=82933 Win=2097120 Len=0
0.928000	129.132.202.24	10.0.0.20	TCP	http	63875	5054	[TCP segment of a reassembled PDU]
0.956000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [ACK] Seq=4023139344 Ack=87933 Win=2097120 Len=0
0.966000	129.132.202.24	10.0.0.20	TCP	http	63875	5054	[TCP segment of a reassembled PDU]
0.991000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [ACK] Seq=4023139344 Ack=92933 Win=2097120 Len=0
1.002000	129.132.202.24	10.0.0.20	TCP	http	63875	5054	[TCP segment of a reassembled PDU]
1.027000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [ACK] Seq=4023139344 Ack=97933 Win=2097120 Len=0
1.038000	129.132.202.24	10.0.0.20	TCP	http	63875	5054	[TCP segment of a reassembled PDU]
1.071000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [ACK] Seq=4023139344 Ack=102933 Win=2097120 Len=0
1.082000	129.132.202.24	10.0.0.20	TCP	http	63875	5054	[TCP segment of a reassembled PDU]
1.109000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [ACK] Seq=4023139344 Ack=107933 Win=2097120 Len=0
1.120000	129.132.202.24	10.0.0.20	TCP	http	63875	5054	[TCP segment of a reassembled PDU]
1.146000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [ACK] Seq=4023139344 Ack=112933 Win=2097120 Len=0
1.155000	129.132.202.24	10.0.0.20	TCP	http	63875	5054	[TCP segment of a reassembled PDU]
1.180000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [ACK] Seq=4023139344 Ack=117933 Win=2097120 Len=0
1.191000	129.132.202.24	10.0.0.20	TCP	http	63875	5054	[TCP segment of a reassembled PDU]
1.214000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [ACK] Seq=4023139344 Ack=122933 Win=2097120 Len=0
1.224000	129.132.202.24	10.0.0.20	TCP	http	63875	5054	[TCP segment of a reassembled PDU]
1.251000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [ACK] Seq=4023139344 Ack=127933 Win=2097120 Len=0
1.261000	129.132.202.24	10.0.0.20	HTTP	http	63875	4934	HTTP/1.1 200 OK (JPEG JFIF image)
1.284000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [ACK] Seq=4023139344 Ack=132813 Win=2097120 Len=0
1.297000	129.132.202.24	10.0.0.20	TCP	http	63877	5054	[TCP segment of a reassembled PDU]
1.322000	10.0.0.20	129.132.202.24	TCP	63877	80	54	63877 > http [ACK] Seq=292529057 Ack=8374 Win=2097120 Len=0
1.332000	129.132.202.24	10.0.0.20	HTTP	http	63877	4424	HTTP/1.1 200 OK (JPEG JFIF image)
1.355000	10.0.0.20	129.132.202.24	TCP	63877	80	54	63877 > http [ACK] Seq=292529057 Ack=12744 Win=2097120 Len=0
1.365000	129.132.202.24	10.0.0.20	TCP	http	63879	5054	[TCP segment of a reassembled PDU]
1.388000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935882 Ack=50342 Win=2097120 Len=0
1.397000	129.132.202.24	10.0.0.20	TCP	http	63879	5054	[TCP segment of a reassembled PDU]
1.424000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935882 Ack=55342 Win=2097120 Len=0
1.434000	129.132.202.24	10.0.0.20	TCP	http	63879	5054	[TCP segment of a reassembled PDU]
1.463000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935882 Ack=60342 Win=2097120 Len=0
1.472000	129.132.202.24	10.0.0.20	TCP	http	63879	5054	[TCP segment of a reassembled PDU]
1.500000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935882 Ack=65342 Win=2097120 Len=0
1.506000	129.132.202.24	10.0.0.20	TCP	http	63879	2726	[TCP segment of a reassembled PDU]
1.521000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935882 Ack=68014 Win=2097120 Len=0
1.533000	129.132.202.24	10.0.0.20	TCP	http	63879	5054	[TCP segment of a reassembled PDU]
1.560000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935882 Ack=73014 Win=2097120 Len=0
1.571000	129.132.202.24	10.0.0.20	TCP	http	63879	5054	[TCP segment of a reassembled PDU]
1.598000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935882 Ack=78014 Win=2097120 Len=0
1.607000	129.132.202.24	10.0.0.20	TCP	http	63879	5054	[TCP segment of a reassembled PDU]
1.635000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935882 Ack=83014 Win=2097120 Len=0
1.645000	129.132.202.24	10.0.0.20	TCP	http	63879	5054	[TCP segment of a reassembled PDU]
1.672000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935882 Ack=88014 Win=2097120 Len=0
1.683000	129.132.202.24	10.0.0.20	TCP	http	63879	5054	[TCP segment of a reassembled PDU]
1.709000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935882 Ack=93014 Win=2097120 Len=0
1.719000	129.132.202.24	10.0.0.20	TCP	http	63879	5054	[TCP segment of a reassembled PDU]
1.746000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935882 Ack=98014 Win=2097120 Len=0
1.756000	129.132.202.24	10.0.0.20	TCP	http	63879	5054	[TCP segment of a reassembled PDU]
1.783000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935882 Ack=103014 Win=2097120 Len=0
1.792000	129.132.202.24	10.0.0.20	TCP	http	63879	5054	[TCP segment of a reassembled PDU]
1.818000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935882 Ack=108014 Win=2097120 Len=0
1.824000	129.132.202.24	10.0.0.20	HTTP	http	63879	818	HTTP/1.1 200 OK (JPEG JFIF image)
1.934000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935882 Ack=108778 Win=2097120 Len=0
31.351000	10.0.0.20	129.132.202.24	TCP	63873	80	54	63873 > http [FIN ACK] Seq=2747000659 Ack=13404 Win=2097120 Len=0
31.360000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [FIN ACK] Seq=4023139344 Ack=132813 Win=2097120 Len=0
31.653000	10.0.0.20	129.132.202.24	TCP	63873	80	54	63873 > http [FIN ACK] Seq=2747000659 Ack=13404 Win=2097120 Len=0

Time	Source	Destination	Proto	Src Port	Dest Port	Length	Info
31.665000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [FIN ACK] Seq=4023139344 Ack=132813 Win=2097120 Len=0
32.055000	10.0.0.20	129.132.202.24	TCP	63873	80	54	63873 > http [FIN ACK] Seq=2747000659 Ack=13404 Win=2097120 Len=0
32.066000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [FIN ACK] Seq=4023139344 Ack=132813 Win=2097120 Len=0
32.434000	10.0.0.20	129.132.202.24	TCP	63877	80	54	63877 > http [FIN ACK] Seq=292529057 Ack=12744 Win=2097120 Len=0
32.449000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [FIN ACK] Seq=947935882 Ack=108778 Win=2097120 Len=0
32.667000	10.0.0.20	129.132.202.24	TCP	63873	80	54	63873 > http [FIN ACK] Seq=2747000659 Ack=13404 Win=2097120 Len=0
32.676000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [FIN ACK] Seq=4023139344 Ack=132813 Win=2097120 Len=0
32.742000	10.0.0.20	129.132.202.24	TCP	63877	80	54	63877 > http [FIN ACK] Seq=292529057 Ack=12744 Win=2097120 Len=0
32.760000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [FIN ACK] Seq=947935882 Ack=108778 Win=2097120 Len=0
33.147000	10.0.0.20	129.132.202.24	TCP	63877	80	54	63877 > http [FIN ACK] Seq=292529057 Ack=12744 Win=2097120 Len=0
33.165000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [FIN ACK] Seq=947935882 Ack=108778 Win=2097120 Len=0
33.546000	129.132.202.24	10.0.0.20	TCP	http	63873	54	http > 63873 [FIN ACK] Seq=13404 Ack=2747000660 Win=65535 Len=0
33.550000	129.132.202.24	10.0.0.20	TCP	http	63875	54	http > 63875 [FIN ACK] Seq=132813 Ack=4023139345 Win=65535 Len=0
33.560000	10.0.0.20	129.132.202.24	TCP	63873	80	54	63873 > http [ACK] Seq=2747000660 Ack=13405 Win=2097120 Len=0
33.565000	10.0.0.20	129.132.202.24	TCP	63875	80	54	63875 > http [ACK] Seq=4023139345 Ack=132814 Win=2097120 Len=0
33.754000	10.0.0.20	129.132.202.24	TCP	63877	80	54	63877 > http [FIN ACK] Seq=292529057 Ack=12744 Win=2097120 Len=0
33.771000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [FIN ACK] Seq=947935882 Ack=108778 Win=2097120 Len=0
34.614000	129.132.202.24	10.0.0.20	TCP	http	63877	54	http > 63877 [FIN ACK] Seq=12744 Ack=292529058 Win=65535 Len=0
34.618000	129.132.202.24	10.0.0.20	TCP	http	63879	54	http > 63879 [FIN ACK] Seq=108778 Ack=947935883 Win=65535 Len=0
34.627000	10.0.0.20	129.132.202.24	TCP	63877	80	54	63877 > http [ACK] Seq=292529058 Ack=12745 Win=2097120 Len=0
34.631000	10.0.0.20	129.132.202.24	TCP	63879	80	54	63879 > http [ACK] Seq=947935883 Ack=108779 Win=2097120 Len=0

Table B.3: This table shows the captured traffic of Test 3. This test shows that it is possible to use parallel HTTP requests to get the content of a website containing text and multiple pictures. The table shows two IP addresses. IP Address 10.0.0.20 is the IP address used by the virtual interface and 129.132.202.24 is the IP address of the target server. There are also TCP packets with a length of 5054 bytes. These are parts of a big TCP segment, which was split by the custom-built TCP stack. The note 'TCP segment of a reassembled PDU' means that the different parts of the TCP segment have to be reassembled in order to obtain the entire TCP segment.

Bibliography

[1] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel and Anmol N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10), 2010.

[2] Manuel Egele, Christopher Kruegel, Engin Kirda and Giovanni Vigna. PiOS: Detecting Privacy Leaks in iOS Applications. Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA. January 2011.

[3] Yuvraj Agarwal and Malcolm Hall. ProtectMyPrivacy: Detecting and mitigating Privacy Leaks on iOS Devices Using Crowdsourcing. MobiSys 2013 – Proceedings of the 11th International Conference on Mobile Systems, Applications and Services, 2013.

[4] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, David Choffnes. Recon: Revealing and Controlling PII Leaks in Mobile Network Traffic. MobiSys 2016 – Proceedings of the 11th International Conference on Mobile Systems, Applications and Services, 2016.

[5] Abbas Razaghpanah, Narseo Valina-Rodriguez, Srikanth Sundaresan, Christian Kreibich, Phillipa Gill, Mark Allman and Vern Paxson. Haystack: In Situ Mobile Traffic Analysis in User Space. 2016.

[6] Statistica. Global mobile OS market share in sales to end users from 1st quarter 2009 to 1st quarter 2016. `http://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems`. Accessed: 28.07.2016

[7] CERT/CC Blog, Will Dormann. The Risks of SSL Inspection. `https://insights.sei.cmu.edu/cert/2015/03/the-risks-of-ssl-inspection.html`. 13.03.2015. Accessed: 27.07.2016

[8] Techcrunch, Kate Conger. Apple will require HTTPS connections for iOS apps by the end of 2016. `https://techcrunch.com/2016/06/14/apple-will-require-https-connections-for-ios-apps-by-the-end-of-2016`. 14.06.2016. Accessed: 27.07.2016

[9] Tim Cook. Apple's commitment to your privacy `http://www.apple.com/privacy`. Accessed: 28.07.2016

[10] Apple. iOS Security White Paper. Page 4,19,20,21,59. 2016.

[11] Stackoverflow. Programmatically obtaining the IMEI or UDID of an iOS Device. `http://stackoverflow.com/questions/31075049/programmatically-obtaining-the-imei-or-udid-of-an-ios-device`. 26.06.2015. Accessed: 28.07.2016

[12] Apple. Privacy Policy. `http://www.apple.com/legal/privacy/en-ww/`. Accessed: 28.07.2016

[13] iOS Developer Library. Network Extension Framework. `https://developer.apple.com/library/ios/documentation/NetworkExtension/Reference/Network_Extension_Framework_Reference`. 21.03.2016. Accessed: 27.07.2016

[14] iOS Developer Library. NEPacketTunnelProvider. https://developer.apple.com/library/ios/documentation/NetworkExtension/Reference/NEPacketTunnelProviderClassRef/index.html#//apple_ref/occ/cl/NEPacketTunnelProvider. 01.03.2016. Accessed: 27.07.2016

[15] Apple. Choose your new iPhone 6. http://www.apple.com/us_smb_78313/shop/buy-iphone/iphone6. Accessed: 30.07.2015

[16] Chip. Home Server: Flexibler Profi fÃ¼r zu Hause. http://www.chip.de/artikel/Speicher-mit-Mehrwert-Festplatten-NAS-Home-Server-6_43953782.html. 17.08.2010. Accessed: 30.07.2015

[17] Mitmproxy. https://mitmproxy.org. Accessed: 31.7.2016

[18] iTunes Preview. Apple Configurator 2. https://itunes.apple.com/ch/app/apple-configurator-2/id1037126344?mt=12. 16.05.2016. Accessed: 20.08.2016

[19] Mitmproxy Documenation. Introduction. http://docs.mitmproxy.org/en/stable/. Accessed: 31.07.2016

[20] Strongswan. The OpenSource IPsec-based VPN Solution. https://www.strongswan.org. 17.04.2016. Accessed: 31.07.2016

[21] Strongswan. strongSwan - About. https://www.strongswan.org/about.html. 09.09.2015. Accessed: 31.07.2016

[22] Pandas. Python Data Analysis Library. http://pandas.pydata.org. Accessed: 20.08.2016

[23] iTunes Preview. Magellan Active App. https://itunes.apple.com/de/app/magellan-active-app/id980756767?mt=8. 21.01.2016. Accessed: 16.08.2016

[24] Seaborn. Seaborn: Statistical data visualization. https://stanford.edu/~mwaskom/software/seaborn/index.html. Accessed: 21.08.2016

[25] Wikipedia. Kernel Density Estimation. https://en.wikipedia.org/wiki/Kernel_density_estimation. Accessed: 21.08.2016

[26] Github. SwiftyJSON. https://github.com/SwiftyJSON/SwiftyJSON. Accessed: 31.08.2016

[27] Github. SwiftyJSON License. https://github.com/SwiftyJSON/SwiftyJSON/blob/master/LICENSE. 6.08.2014. Accessed: 1.09.2016

[28] Github. Daniel Gindi: Charts. https://github.com/danielgindi/Charts. Accessed: 21.08.2016

[29] Github. CocoaAsyncSocket. https://github.com/robbiehanson/CocoaAsyncSocket. Accessed: 31.08.2016

[30] Github. CocoaAsyncSocket License. https://github.com/robbiehanson/CocoaAsyncSocket/wiki/License. Accessed: 1.09.2016

[31] Wireshark. About Wireshark. https://www.wireshark.org. Accessed: 15.08.2016

[32] iTunes Preview. Dolphin Internet Browser. https://itunes.apple.com/ch/app/dolphin-internet-browser-schnell/id452204407?mt=8. 25.07.2016. Accessed: 28.08.2016

[33] iTunes Preview. Speedtest.pro Speed Test and WiFi Finder. https://itunes.apple.com/ch/app/speedtest.pro-speed-test-wifi/id616145031?mt=8. 17.05.2016. Accessed: 28.08.2016

[34] TestPlant. eggPlant Mobile. http://www.testplant.com/eggplant/testing-tools/eggplant-mobile-eggon/. Accessed: 23.08.2016