University of Passau
Faculty of Computer Science and Mathematics

Chair of IT Security
Prof. Dr. Joachim Posegga

Master's Thesis
in
Computer Science

Improving Smartphone Privacy with a Privacy Proxy

Florentin Wieser
100465

Date:	2022-04-22
Supervisors:	Prof. Dr. Joachim Posegga
	Prof. Dr. Hans P. Reiser
Advisor:	Dr. Henrich C. Pöhls MSc.Info.-Sec.

Wieser, Florentin
Waldesruhe 6
81377, München

ERKLÄRUNG

Ich erkläre, dass ich die vorliegende Arbeit mit dem Titel „Improving Smartphone Privacy with a Privacy Proxy“ selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe und dass alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind.

Mit der aktuell geltenden Fassung der Satzung der Universität Passau zur Sicherung guter wissenschaftlicher Praxis und für den Umgang mit wissenschaftlichem Fehlverhalten vom 31. Juli 2008 (vABlUP Seite 283) bin ich vertraut.

Ich erkläre mich einverstanden mit einer Überprüfung der Arbeit unter Zuhilfenahme von Dienstleistungen Dritter (z.B. Anti-Plagiatssoftware) zur Gewährleistung der einwandfreien Kennzeichnung übernommener Ausführungen ohne Verletzung geistigen Eigentums an einem von anderen geschaffenen urheberrechtlich geschützten Werk oder von anderen stammenden wesentlichen wissenschaftlichen Erkenntnissen, Hypothesen, Lehren oder Forschungsansätzen.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

.....
(Name, Vorname)

Translation of German text (notice: Only the German text is legally binding)

I hereby confirm that I have composed the present scientific work entitled “Improving Smartphone Privacy with a Privacy Proxy” independently without anybody else’s assistance and utilising no sources or resources other than those specified. I certify that any content adopted literally or in substance has been properly identified and attributed.

I have familiarised myself with the University of Passau’s most recent Guidelines for Good Scientific Practice and Scientific Misconduct Ramifications of 31 July 2008 (vABlUP Seite 283).

I declare my consent to the use of third-party services (e.g. anti-plagiarism software) for the examination of my work to verify the absence of impermissible representation of adopted content without adequate attribution, which would violate the intellectual property rights of others by claiming ownership of somebody else’s work, scientific findings, hypotheses, teachings or research approaches.

This thesis has not been submitted to any other examination authority in the same or a similar form.

Supervisor contacts:

Prof. Dr. Joachim Posegga
Chair of IT Security
Universität Passau
Email: jp@sec.uni-passau.de
Web: https://www.sec.uni-passau.de

Prof. Dr. Hans P. Reiser
Department of Computer Sciences
Reykjavik University
Iceland
Email: hans.reiser@uni-passau.de

Advisor contacts:

Dr. Henrich C. Pöhls MSc.Info.-Sec.
Chair of IT Security
Universität Passau
Email: hp@sec.uni-passau.de
Web: https://henrich.poehls.com/

Abstract

Privacy and privacy-related issues appear regularly in public discussions. Recently, they even pushed providers of mobile operating systems to introduce a new privacy feature which restricts access to an advertising identifier. This feature affects advertising companies which identify users across multiple applications. However, the remaining communication originating from apps is still unaffected by the new feature. Here, data that could have implications on privacy still could be transmitted. Proposed solutions to increase privacy often require deep technical understanding like jailbreaking a smartphone. To offer a privacy enhancing technology to less tech-savvy users, a proxy-based solution is designed and implemented. The privacy proxy acts as a man-in-the-middle and inspects the HTTP traffic sent by smartphones. When privacy-related data is found, this data should be obfuscated by the proxy. An HTTP request could contain data in multiple locations. Three locations are considered by the privacy proxy: headers, the body and URL parameters. Different data may have different implications on privacy. To do justice to this fact, each single data point is manually classified on the potential impact on privacy it may have. Based on said classification, the proxy decides whether to obfuscate the respective data. To evaluate the impact on privacy of the proposed approach, data is collected for six example apps (three each from categories *News* and *Weather*). For each app, three modes of operation were compared: without any interference of the proxy, with blocking connections to known tracking- & advertising providers and with said implementation of the obfuscation mechanism. Also, the increase of privacy achieved by incorporating a cache at the privacy proxy was examined. Results show, that indeed the number of potential privacy leaks decreases with the usage of additional mechanisms. However, it was necessary to allow the leak of some data points which were previously considered as privacy impacting in order to retain app functionality.

Contents

1 Introduction

1.1 Motivation

A survey performed by the Pew Research Center [9], found that 70% of American adults say their personal data is less secure than five years ago. An even larger percentage (81%) says that the potential risk they face because of data collection by companies outweighs the benefits. These trends even have effects on big tech companies. Apple recently introduced a function that restricts access to the advertising identifier and only reveals it when the user explicitly allows tracking [6]. According to Flurry, a widespread app analysis platform, the majority of users decides not to opt-in for tracking [37]. Google, on whose services many apps of the widespread operating system Android rely, also announced a similar feature which allows users to opt-out of interest-based advertising [27].

Losing access to advertising IDs makes it harder for advertising companies to identify users across apps. Service providers, which supply content the user intends to reach, and advertising companies still can receive lots of other data from apps: personal information, specific device information or location data. Valuable service usage (from an end user's point of view) might require some of that data – for example the user's language for news articles. However, external parties also might receive data that is either not required (e.g. for analysis services) or data that is sufficient in lower resolution (e.g. location data for weather forecast) for helpful service usage. Cutting off non-essential data could get one closer to the data minimization principle. According to Kaaniche, Laurent and Belguith this principle is a fundamental feature of privacy preservation [35].

This reduction of the external data footprint could be achieved by inspecting and interfering with the communication between the smartphone and the internet. Said intervention would not only increase personal privacy, but could also mitigate the aftermath of data breaches – which have notably increased over the last decade (see figure A.1 in appendix) [28].

1.2 Goals

Main Goals

The main goal of this thesis is to implement and evaluate an HTTP proxy which should increase the privacy of smartphone users. Every HTTP request contains multiple data points for example in headers or in the body. Each data point could contain privacy impacting data (e.g. AccountIDs). Therefore, the amount of privacy impacting data points is used as

a metric for the evaluation of different approaches.

Baseline for the evaluation is the amount of data points that are transmitted when no interference by the proxy is performed at all. Against this "ground truth", two methods of privacy enhancement should be assessed. The first mechanism checks the target domain of each encountered request with a list of known tracking and analytics (T&A) domains and blocks the request if a match occurs. The second approach should work on a more granular level: on the actual content of requests. In addition to blocking known T&A servers, it should reduce the amount of privacy impacting data points sent out. This should be performed by automatically obfuscating data points which are considered privacy impacting. Obfuscation (or synonymous anonymization) hereby means the replacing or blurring of data.

Besides the just mentioned methods that interfere with traffic, also a caching approach should be assessed. Here, the extent to which caching is possible in the mobile app context and the resulting effects on privacy should be examined.

An aspect that must be maintained in all approaches is the functionality of apps. For this purpose, a *core functionality* is defined for each app and this core functionality must remain intact when applying the different privacy enhancing approaches.

Soft Goals

Besides the main goals, also a soft goal[1] is defined. It states that the resulting solution should be easy to set up for an end user. In particular, deep technical understanding should not be necessary to use the privacy proxy and techniques like jailbreaking should be avoided.

Core Assumption

The core assumption used for this thesis is, that the number of leaked data points corresponds to the extent of privacy loss. As not every data point contains privacy impacting information, data points are divided into different categories – based on the potential impact on privacy they oppose.
Other work compares privacy protecting mechanisms in terms of the number of HTTP requests [40], but this criterion is pretty coarse and it ignores the actual content of HTTP requests.

1.3 Outline

First, the terms used in this thesis are defined in section 2. Chapter 3 describes the current state of privacy-related topics in the smartphone context. Particular attention is paid to the role of operating system providers. In chapter 4, technical information on the employed attack and an attribution method is given. Also, the threat model is described. Chapter 5 provides information about approaches proposed by other researchers to increase privacy in the smartphone context. Next, chapter 6 explains the solution designed for this thesis. Also, the evaluation scheme will be outlined. Afterwards, the technical implementation details are shown. Results of the evaluation and its discussion are provided in chapter 8. Finally, a conclusion is provided and suggestions for future work in this field are given in chapter 9.

[1] Soft Goals: objectives that do not have clear-cut criterion for their satisfaction [43]

2

Terms & Definitions

This chapter defines terms and definitions that are used throughout this thesis.

Personal information (PI) is defined by the European commission as "any information that relates to an identified or identifiable living individual" [20]. This data includes (among other things) names, addresses, location data, IP-addresses and advertising identifiers.
In this thesis, the definition is extended by data that could potentially be used to identify users via the fingerprinting method. Fingerprinting combines multiple parameters into a single one, which then could be used to identify devices.

Service providers (or first parties) are the content suppliers that users intend to reach [40]. For weather apps, e.g. they provide the weather data and means to retrieve them (in this case mobile applications). **Third parties** can be advertisers, ad exchanges or other actors that provide services to first parties (such as web analytics) [40]. **External parties** encompass both service providers and third parties.

The Cambridge dictionary defines **privacy** as "someone's right to keep their personal matters and relationships secret" [11]. In the context of the privacy proxy, the secret-keeping is performed by avoiding that service providers or third parties receive PI.

HTTP traffic consists of requests and responses. A pair of a request and a response is referred to as a **flow** in this thesis.

The data that is transmitted via HTTP is often structured. Most of the structures are in the form of **key-value** pairs or may easily be transformed into them. A **data point** refers to a granular point of information – e.g. one isolated key-value pair.

Data leaks refer to all actions in which data (that could have privacy implications) is transferred to either a service provider or a third party employed by the service provider.

"A **tracking pixel** [...] is a 1×1 pixel graphic used to track user behavior, site conversions, web traffic, and other metrics similar to a cookie" ([14]).

3 Current Position of Privacy in Smartphone Context

In this chapter, the current standpoint of privacy-related topics in the smartphone context is described. Especially the measures implemented by major operating system (OS) providers are briefly depicted.

3.1 Permission Management Systems

Smartphone OS providers[2] already facilitate some kind of privacy protection by implementing permission management systems. App developers have to declare which data/resources their app wants to access. This information is shown to the user either at installation time or at run-time. Then, the user can make a decision whether to grant the required permissions to the respective app. Figure 3.1 shows an example popup asking for the location of the smartphone. Also, the feature "precise location" is visible. It is a rather new[3] feature and allows the user to decide whether accurate or inaccurate location data should be revealed to requesting apps. Other examples for permissions include access to contact lists, the ability to send push notifications and access to data stored on the phone [5].

However, these systems have some problematic characteristics. One point is that apps may request permissions not required for their core functionality [57]. In their development guidelines, the OS providers recommend data minimization [26] and to only request permissions when the app clearly needs access to the data/resource [4]. Yet, the developer is free to choose which permissions are required and according to Sha and Liu [59], there is a lack of strict development specifications and effective supervision. This leads to the problem of overprivileged apps, which describes that apps request and gain access to permissions which are not needed for their execution [44]. Research on this topic found, that developers regularly choose to over-privilege their apps [44] [59].

Another issue with permission management systems is how users react to permission requesting popups. Often the user is forced to give all necessary permissions in order to use an app without potentially losing some or all functionality [21]. Or users just blindly accept all prompts because the different permission requests look alike and such popups occur regularly [47]. These problems undermine the effectiveness of such permission management systems.

[2] Google for Android, Apple for iOS
[3] This feature was introduced in iOS 14 which was released in 2020 [39]

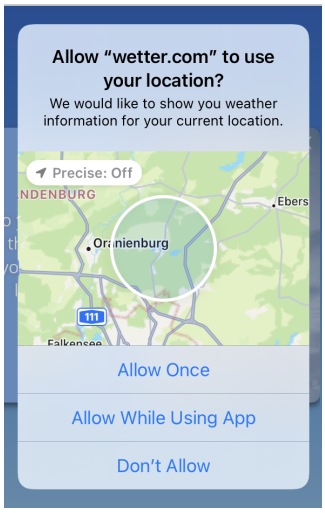


Figure 3.1: Example popup from the permission management system. Also visible: the "precise location" feature to specify the accuracy of location data

3.2 Handling of Developers

Further risk to the privacy of smartphone users is posed by the way OS providers deal with developers. Van Hoboken and Fathaigh evaluated smartphone platforms as privacy regulators [57]. According to their findings, OS providers make developers sign contracts to comply with privacy laws and other rules (e.g. respectful usage of unique identifiers or rules for device identification).
Notwithstanding this, the authors also found that the enforcement of these policies is a non-trivial task. The platforms can't investigate each app manually due to the mass of apps and can't monitor what developers do with collected data. However, they review certain apps (e.g. after hints of rule violations) and remove the app from the app store or ban the developer account if policies actually were ignored.

Summary

In summary, the privacy protection mechanisms employed today by smartphone OS providers are not sufficient. Permission management systems are outplayed by developers who overprivilege their apps. Violations against guidelines or contracts are hard to detect with the amount of available apps and the actions taken by OS providers are rather reactive than proactive. This also exposes that developers are only slightly restricted in their design decisions. They have to move around technical specifications dictated by OS providers (like asking for permission to access the location), but other than that they can freely choose what data to collect and where to transmit it to.
As this is not expected to change in the near future, external solutions that increase privacy are required.

4 Background

In this section, some background information on the employed attack and an attribution method to identify devices called fingerprinting is given. Finally, the threat model considered for the thesis is described.

4.1 Man-in-the-Middle Attack

According to the *Google Transparency Report*, over 90% of pages loaded in Chrome use HTTPS[4] on the platforms Windows, Android and Mac [25]. This shows, that the majority of HTTP-traffic is encrypted nowadays. While encryption is generally desired by users, it poses a drawback for traffic analyzing and interfering tasks. In order to access the plaintext traffic, one has to strip the encryption and re-encrypt the traffic after the desired interactions.

An approach to do so is called a man-in-the-middle attack. In this approach, the attacker interposes in the communication between the client and the server. Towards the client it masquerades as the target server and towards the target server it acts like a client. After initialization, there are now two encrypted channels (client - attacker AND attacker - server) instead of just one (client - server) in an unaffected setup. The attacker now has access to the (decrypted) traffic.

A counter measure to this attack is the Certificate Authority (CA) system:

> [...] [T]he Certificate Authority system is designed to prevent exactly this attack, by allowing a trusted third-party to cryptographically sign a server's certificates to verify that they are legit. If this signature doesn't match or is from a non-trusted party, a secure client will simply drop the connection and refuse to proceed. ([41])

The trust of these third parties is guaranteed by so called root certificates. These root certificates are pre-downloaded and shipped with operating systems or specific applications [8].
If the middleman is not trusted, the client would just drop connections to it. To circumvent this issue, the middleman can act as a CA itself. The respective root certificate of the interposed CA then could be installed on the client. After this is done, the client recognizes the signatures provided by the middleman as valid and allows to communicate with it.

[4]HTTPS is the secure version of HTTP. It uses the TLS protocol to encrypt communication [13]

4.2 Mobile Device Fingerprinting

Fingerprinting is an attribution method that combines available parameters such as the device name, device type, OS version, IP address, carrier or other attributes to form a digital fingerprint which allows to recognize a device [46]. By combining many data points (which by themselves reveal very little about a device), a rather specific identifier could be built (even though not 100% deterministic).

This technique has been around in website tracking and Mozilla's Firefox blocks fingerprinting resources since quite some time[5]. While there clearly exists evidence about the use of this technique in the smartphone context [34] [46], it hasn't gained much attention and solutions to block fingerprinting are absent.

Nevertheless, this tracking technique should not be ignored. Especially the lack of visibility (fingerprinting could be done with accessible data points which don't require a permission by the user) make it a privacy-related technique. The privacy proxy should do justice to fingerprinting by also considering data points suitable for fingerprints as privacy impacting. Subsequent, these data points also should be obfuscated.

4.3 Threat Model

To specify and clarify the objective of the thesis, a threat model is described.

Basically, there are four parties involved in the considered scenario:

- **Client**: Smartphone which runs the desired apps
- **Privacy Proxy**: The intermediate proxy which is subject of this thesis. Tunnels communication from the client to increase his privacy
- **Service Providers**: The party which provides the content the client wants to access
- **Third Parties**: Other entities which are employed by service providers like analytics services or advertising platforms

It is assumed that the service provider as well as third parties are honest but curious.

The trust status of the privacy proxy depends on the way it is hosted. When self-hosted, no other party has to be trusted. When the proxy is provided by another party, this party has to be trusted (as it gains access to the decrypted traffic).
However, a soft goal of this thesis is the ease of setting up the system. It allows that the proxy also could be deployed separately for multiple users with limited effort. This would eliminate the need to trust a further entity.

Next, the asset at risk is named, as well as the potential threat.

Asset: The main asset that should be protected is PI of the client – especially data that is not necessarily required for accessing the desired content. By also considering data points suitable for fingerprinting, the client's privacy is increased by the reduction of his uniqueness. This impedes non-deterministic tracking methods.

Threats: The threat the privacy proxy should mitigate is the potential misuse of data by service providers or third parties. Once they obtained data, the user has no control over the data and its usages anymore. When reducing the transmitted data to the bare minimum required to access the service, this would also reduce the abilities of service providers and third parties to perform further actions on the data (e.g. tracking).

[5] Firefox blocks fingerprinting resources since version 72 which was released in January 2020 [19]

5 Related Work

The following gives an overview of privacy enhancing technologies (PET) suggested by other researchers. In particular, a look at blocking, caching and obfuscating concepts is taken.

According to the taxonomy of PET proposed by Kaaniche et al. [35], PETs can be classified into three groups: user side, channel side and server side techniques.
User side techniques require actions by end-users to protect their privacy, like installing anti-tracking software. Channel side mechanisms interact with privacy properties of the communication channel between servers and end-users. Techniques considered as server sided demand the service provider to incorporate additional processing steps to increase user's privacy. As this work intends to increase privacy against service providers, server sided techniques will not be discussed here.

5.1 Blocking Information Flow

First, two examples for the most straight-forward approach are described. By suppressing the transmission of PI to locations outside the device, the privacy of a smartphone user is improved. This approach can be implemented either on the device (making it a user side PET) or on a separate proxy (making it a channel side PET).

User Side Approach

The blocking itself can further be partitioned: either by completely restricting internet access to certain applications or by blocking all connections to hostnames[6] which are contained in a predefined list [47].

One implementation that allows to arbitrarily restrict the ability of applications to access the internet is *IAFWall+*[7]. It basically is an advanced editor for the linux firewall *iptables* with which one could manage the ability of apps to access the internet. While the approach may work for simple applications like alarm clocks, it can't improve user's privacy in more complex apps that require to access regularly changing data (like weather forecasts).

An example for blocking with the help of hostname lists is *AdAway*[8]. It works by leveraging the *hosts* file of a system. This file associates IP addresses with hostnames [38]. *AdAway*

[6] hostnames is used as a synonym for domain names in this thesis
[7] `https://github.com/ukanth/afwall`
[8] `https://adaway.org`

loads community managed lists of known advertising hostnames into that file and associates each hostname with the IP address of localhost [1]. This redirection of requests to the localhost then makes the requests fail.
One crunch point when using hostname lists is to find the right amount of hostnames that should be blocked: either one accepts data leakage (when leaving out hostnames) or one may risk service functionality (when including to many).

Channel Side Approach

The previously explained approach of blocking lists of known advertising hostnames can also be employed at the network side. A famous example is *pi-hole*[9]. It employs the same mechanism and provides the blocking technique to all devices that use it as a DNS resolver.

5.2 Caching Applications

A completely different concept bases on caching. When a request is answered with a cached response (and thus is not forwarded to the service provider), the service provider receives less PI. In the caching context, one also speaks of cache hits (when a response was already saved) and cache misses (when the response was not present in the cache). In the following, an approach working on smartphones (user side technique) and a concept working on the channel side are explained.

User Side Approach

On the user side, two different strategies are described. Both employ caching at the smartphone itself, but the method of getting content into the cache differs. In the first approach, the cache tries to fetch potentially interesting content before the user accesses it. In the second solution, content is actively pushed to the smartphone from the outside.

Prefetching Cache

A proposal for a prefetching cache was made by Amini et al. with their system *Caché* [2]. Their main idea is to periodically fetch location-enhanced content for a previously specified area of interest (e.g. a city) and then cache the received data on the phone. When an app wants to access the location, *Caché* serves the prefetched data.

The increase of privacy against service providers hereby is achieved on two dimensions: coarsening of location data and the introduction of a temporal shift.
Coarsening concerns the accuracy of location data. Instead of receiving exact coordinates, the service provider just gains knowledge about the general area the user is interested in.
The second dimension is the temporal shift. As the requests for location enhanced content are done in advance, the service provider doesn't know about the timing of accessing such content.

An interesting aspect of *Caché* is the integration with other applications, as it doesn't require access to underlying OS mechanisms. It is basically an Android service which runs in the background and takes care of the caching process. Apps that should use *Caché*, must be modified to redirect HTTP request to the caching service. The required modifications are rather small, but require app developers to enhance their app. If the app developers are not willing to do this extra work, *Caché* is not able to increase the privacy for those apps.

[9] `https://pi-hole.net`

Cache Pushing

A caching solution that operates by proactively pushing content to the phone is suggested by Chen et al. in their work "Preserving Location Privacy based on Distributed Cache Pushing" [12]. Their target is to provide k-anonymity[10] to the location data of users which use location enhanced services. The target is achieved by employing multiple caches distributed in different regions. Each cache stores location-related data and proactively pushes this data to bypassing users. When a user requests location enhanced data, he will check his received cache data for matches. If the requested content is available, the privacy of the user is preserved as the service provider never saw any request. If the content is not available in the cache, the user will contact the service provider. To also guarantee k-anonymity in this case, the user adds $k-1$ distortion requests with random parameters. All k responses from the service provider are then forwarded to the user and the caching system.

Channel Side Approach

Conventional caching approaches operate on the channel side. Usually, they are interposed between the user and the internet. For every request made by a user, a cache checks if it already has the response from the server in his storage. If so, a cache hit occurred and the stored content is replied. If not, the cache missed the request and forwards it to the intended server. The response is saved by the cache and then forwarded to the user.

There are also techniques in place that limit caching. For example, in the HTTP protocol the *Cache-Control* header is defined [30]. It contains instructions for caches how to handle data associated with this flow (e.g. to not store the content). While some caching software has the ability to circumvent these mechanisms [54], it should be noted that this violates the HTTP standard and therefore can cause issues.

5.3 Obfuscating Data

The last PET works by obfuscating data. Unlike blocking of hostnames, where all requests to certain hostnames are blocked, it allows a more granular differentiation. This is, because it modifies single data points and thus can process each data point differently.

User Side Approaches

An approach which increases privacy through location obfuscation is *LPGuardian* by Fawaz and Shin [22]. It is an Android app which directly interferes with location requests of apps. Based on the app type (weather, fitness tracker), app status (running in background or foreground) and users choice, the location returned could be a dummy location or of low/high granularity. For example, weather forecast apps should work sufficiently with city level granularity while navigation apps require more specific location data. With this approach, Fawaz and Shin increased privacy regarding location data while maintaining the functionality of apps.

In their survey on privacy enforcement in the smartphone context, Pennekamp, Henze and Wehrle presented multiple apps which are quite similar to *LPGuardian*. While they may employ different technical approaches and target different data, they all "[...] require modified system images or root user access as they rely on the modification of application data" ([47]).

[10] k-anonymity "[...] guarantees that in a set of k individuals, the identity of each one cannot be disclosed from at least k-1 individuals" ([15])

Channel Side Approaches

One solution to improve privacy without the requirement of the user being technically versed, is to shift the privacy increasing mechanisms to a proxy server. After a simple setup, the smartphone routes its traffic over said proxy. With this approach, there is no need to jailbreak the target device or to manipulate the firmware of the phone. This approach has been conducted by Rao et al. with their system *Meddle* [48] [49]. Smartphones connect to it via a Virtual Private Network (VPN) and then route their traffic through it. *Meddle* is a platform that allows plugins (so called software middleboxes) to control (intercept, block, modify) the traffic flowing through it. One feature included in such a middlebox is the blocking of connections to known advertising services.

ReCon [50] is a middlebox which builds ontop *Meddle*. It also tries to improve privacy by blurring/hiding PI. The authors use machine learning (ML) to detect and predict privacy impacting information in traffic and then let the user decide whether to block, blur or share said information. In detail, they divide flows into multiple words by splitting it at specific delimiters. Then they apply a "bag-of-words" model[11] to extract features and train with data from controlled experiments. As different destination domains use different encodings, but a single destination often uses the same encoding, one model was trained per domain. These models then were used to classify whether new flows contain PI or not. If they contain PI, the user can choose what happens next. Either the request is blocked, the information is obfuscated or the original data is forwarded unmodified. His decision then is fed back to the ML model via reinforcement.
As *ReCon* shares some functionality with the proposed privacy proxy, a closer look at the differences is taken in chapter 6.2.

[11] "The bag-of-words model [...] is a representation that turns arbitrary text into fixed-length vectors by counting how many times each word appears" ([60])

6 System Design of the Privacy Proxy

The design of the privacy proxy should be similar to the setup of Rao et al. An intermediate entity which could control all traffic sent from the smartphone should be employed. To reduce the external data footprint, the intermediate entity should inspect traffic flowing through it and obfuscate data (e.g. coarse location data or replace IDs with dummy values) if possible.

6.1 Infrastructure

First the layout of the privacy proxy should be explained. Then, a look at the HTTP protocol and a classification of data points on their impact on privacy is taken. Finally, the possible portability of the proxy to the mobile network is described.

6.1.1 Setup

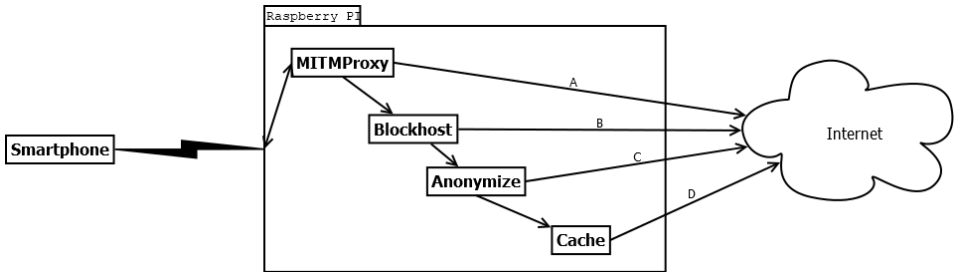


Figure 6.1: Basic setup of *mitmproxy* with four modes of interference: (A) no interference, (B) blocking known T&A hostnames, (C) anonymizing of traffic and (D) caching of traffic

In figure 6.1 the basic setup can be seen. A Raspberry PI will be used as the intermediate server. It will be set up as a WiFi access point to which the target smartphone will connect. On the Raspberry PI, the *mitmproxy*[12] software will be installed. It allows interception, analysis and modification of encrypted web traffic.

[12] `https://mitmproxy.org`

Mode	Name	Details
A	No Interference	Passively inspect traffic
B	Block T&A hosts	Actively block requests to known T&A servers
C	Anonymize	Mode *B* + Replace/Blur data
D	Caching	Mode *C* + Cache app traffic

Table 6.1: Different modes of interference provided by the privacy proxy

In table 6.1 the different modes of interference are described. The first mode *A* hereby describes **no interference** at all. In this case, the traffic is just decrypted by *mitmproxy* and then analyzed.

The next mode *B* performs some interference with the traffic. As the name **blockhost** already suggests, in this step the target host of each request will be cross-checked with a list of known T&A hostnames. If there is a match, the respective request will be blocked.

Mode *C*, **anonymize**, refers to the step where PI leaks in traffic will be obfuscated. Data points which impact privacy should be replaced with hardcoded values. When location data is observed, the resolution should be reduced (meaning a less precise location will be revealed). This mode also includes mode *B* – blocking of known hostnames.

The last mode *D* refers to the **caching** approach. Basically, the task here is to evaluate whether caching is feasible in the mobile app environment and whether a privacy increase could be achieved. The evaluation should be performed on top of mode *C*.

In order to develop the listed modes, knowledge about the HTTP structure and the locations where it could leak data is required. The respective analysis can be found in chapter 6.1.2. Further, different data points may have different implications on privacy. Therefore, a classification on the effect on privacy of different data points is performed in chapter 6.1.3.

6.1.2 Possible Points of Data Transmission in HTTP Protocol

As the apps all use the HTTP protocol to communicate and data leaks happen on the upstream, all points of possible data leakages in HTTP requests should be assessed. According to the HTTP protocol specification, a request contains a request line (containing the URL which may optionally contain a query string), header fields and an optional message body [29]. Therefore, points which may contain data and should be investigated are the URL, headers and the body.

However, one restriction should be introduced for URLs. They consist of two parts: the URL path and an optional query string.
While the query string is not standardized, it is often used to carry key-value information [33]. In this case, one speaks of path parameters or URL parameters. As this information is easy to access, path parameters should be considered by the privacy proxy.
More complex is the consideration of URL paths. It is very hard to distinguish whether a part of the URL path could transfer privacy relevant data or just points to content. Examples for privacy relevant data could be Universal Unique Identifiers (UUIDs) or other identifiers. On the other hand, content management systems or similar applications may store data using UUIDs. During first insights into traffic generated by the evaluated apps, both cases were observed. In table 6.2, two examples can be seen: the first was a request to a tracking pixel (GIF image with size $1x1$ px) located on the URL. The second URL containins multiple UUIDs but returns actual content – the ZDF logo.

As it is very hard to infer the impact on privacy posed by URL paths, the paths were excluded and just URL parameters were considered. However, the occurrence of tracking

No.	URL	Content
1	`https://hr5wdf59xmgurbyaf96vfa222zj581646991791.uaid.nmrodam.com`	Tracking pixel
2	`https://optanon.blob.core.windows.net/logos/f95a8896-6a19-4658-af8e-a796983dd11e/83cf0c1b-c7be-43aa-9e4d-1818ad05b2f5/c81885fd-77d9-43e5-84c0-49deadeffef8/ic_logo.png`	Logo

Table 6.2: Two URLs containing UUIDs. The first URL returns a tracking pixel, the second actual content: a logo

pixels should be considered in the analysis. Here, the focus should be put on the content of the response instead of the URL path of the request.

6.1.3 Classification of Data Points on Privacy Impact

As not every data point of an HTTP request is expected to contain potentially privacy impacting content, some differentiation should be included. A manual classification should be employed to account for this fact. The vast majority of the data is in the form of key-value pairs. Therefore, the name of the key, the datatype of the value and the value itself of each pair should be consulted for the categorization. The categorization should also differentiate on the location of the key-value pair in the HTTP request: in the header, body or in path parameters. The final categorization assigns to each key observed in the *no interference* mode one of the four distinct groups which can be seen in table 6.3.

	Potential Privacy Impact	Description	Examples
0	No Impact	Technical Fields	Encoding, Content-Type, Size, Language, Timestamps
1	Possible Impact	Data Suitable for Fingerprinting, Other IDs	Model, OS Version, Authentication Parameters
2	Suspected Impact	Unique Data	AccountIDs, UUIDs, InstallIDs
3	Geolocation Data	GPS Coordinates	Latitude, Longitude

Table 6.3: Categories used to estimate the privacy impact of data points

6.1.4 Portability to Mobile Networks

So far, the setup only affects devices connected to the local WiFi access point. By tunneling traffic through a VPN connection to the privacy proxy, the privacy enhancing functionality could be employed in mobile networks (also see approach of Rao et al. [48]).
A visualization can be seen in figure 6.2. Smartphone A uses the WiFi connection and smartphone B tunnels its traffic via an encrypted VPN connection to the proxy. The method how smartphone B is connected to the internet is irrelevant. After the privacy proxy performed its privacy enhancing mechanisms, it forwards the traffic to the internet.
This enhancement is basically a virtual extension of the previous setup. The combination of *mitmproxy* with a VPN server has been done previously [18]. Therefore, the transition is not explicitly evaluated in this thesis.

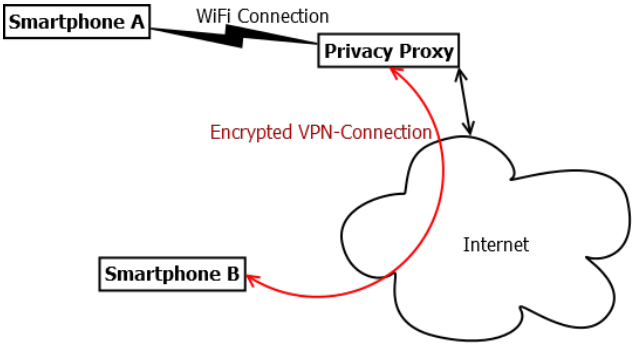


Figure 6.2: Visualization of the current setup and the extension to a VPN. Smartphone *A* uses the proposed privacy proxy setup. Smartphone *B* routes its traffic via an encrypted VPN connection to the privacy proxy which then forwards the traffic to the internet.

6.2 Comparison of Proposed Solution with ReCon

As the system *ReCon* is quite similar to the privacy proxy proposed in this thesis, some differences will be highlighted in the following section.

One issue that should be seen critically is the use of ML. Rao et al. use ML to identify which requests could leak data. While this procedure isn't problematic in general, the usage in combination with encrypted traffic could be. Using ML to build meaningful models requires a lot of data – in *ReCons* case favorably for a broad spectrum of hosts (as they train one model per domain). These amounts and especially the breath of data can best be generated by assessing the traffic of multiple users. This however is the crux: accessing the decrypted traffic of multiple users would mean that they loose the confidentiality encryption offers. If *ReCon* is set up for a single user, the confidentiality issue is mitigated. But it leaves the ML algorithm with a narrow and small sized data set.
In contrast, the privacy proxy proposed in this thesis is not affected by the number of users connected to it. The static categorization is the same for one or multiple users.

As the source code of *ReCon* is available[13], a look at it was taken. As it turned out, only the part which covers the ML process is publicly available. Thus, the classifying models were trained with the data sets that were also used in the evaluation of *ReCon*.
While the training of the models worked without problems, the underlying system composition of *ReCon* is not available and only described very shallow. In their paper, the authors state that they use *SSLsplit* to decrypt traffic, *tcpdump* to dump traffic and *zeek*[14] (formerly named *bro*) to then extract HTTP data [50].

As the exact procedure is not described, some investigation was performed to get the result of the classifier for an example request. An example log entry was taken from *zeek's* documentation, as it showed a quite similar structure to the files available in *ReCons* data set. However, some information is missing in *zeek's* log files. This might be due to interim updates of *zeek* or additional configuration details. When trying to get a classification for the example log file, a `NullPointerException` is thrown.

With this lack of documentation and the crashing classifier, a comparison of the effects of *ReCon* and the privacy proxy proposed by this thesis is unfortunately not feasible.

[13] https://github.com/Eyasics/recon
[14] *zeek* is a passive and open-source network traffic analyzer [55]

PI Labels	IDFA
URL	crwdcntrl.net/5/c=2215/tp=TWCN/ mid=RECON_IDFA_59b6b10871f34...

Figure 6.3: Example excerpt for a correctly labeled request. The IDFA was detected and is present in the URL

PI Labels	None	
URL	waze.com/shields_conf_new_eu	
Headers	X-WAZE-LOCATION	48.186642\x2c16.346910
	...	...

Figure 6.4: Example excerpt for a request that misses the location data inside the header *X-WAZE-LOCATION*. The authors intended to also cover location data

Comparison of Data Classification

Another point for comparison of the proposed privacy proxy with *ReCon* is the data classification. In *ReCon*, the authors consider the following information:

- Device Identifiers (like Ad ID, IMEI, MAC address)
- User Identifiers (e.g. name, gender, date of birth, email address)
- Contact Information (phone numbers, contact books)
- Location (GPS coordinates, zip codes)
- Credentials (username, password)

A flow is considered to contain PI when it contains "[...] the conspicuous PI[...] that we loaded onto devices and used as input to text fields" ([50]). To validate this approach, the classification of the data used as a baseline for the ML algorithm was explored. As this thesis focuses on iOS, also only the iOS data set was examined.

First, it should be noted that *ReCon* works on the same information as the presented privacy proxy: it considers headers, the URL and the body. Manual checks of some samples showed that indeed the classification is correct when previously entered PI is present in the flow – often the IDFA was found and correctly labeled. However, it also was observed that privacy impacting information was present in the flow and absent in the PI labels. This rarely happened, but shows the shortcoming of the automated PI detection. In the following, some classification examples will be shown.

The first example (see figure 6.3) shows a correctly labeled flow. In the URL, the IDFA (Apple's Ad ID) can be seen (complemented with the prefix *RECON_IDFA*).
Next, an example which misses information *ReCon* actually intends to recognize is depicted in figure 6.4. Here, the location data present in a header is omitted and not labeled.
A similar situation can be seen in the last example (figure 6.5). In that case, the *device_id* is part of the URL query yet the flow contains no labels. Also, data which can be used for device fingerprinting is transmitted in the header. While this type of data is not covered by *ReCon*, it still could be used to identify users.

Overall, it can be stated that the labeled data used for the ML algorithm covers PI well, that previously was known. Data that was out of that scope, seemed to be unrecognized which in the end would also affect the quality of the trained models. Besides this issue, other data points which also may have an impact on privacy (like fingerprint components) were not considered in *ReCon*.

PI Labels	None	
URL	liftoff.io/mopub/win_notice?...& device_id_sha1=6c65025faadad4...	
Headers	USER-AGENT	Mozilla/5.0 (iPhone; CPU iPhone OS 8_4_1...
	...	...

Figure 6.5: Example excerpt for a request that misses both: data *ReCons* authors targeted (device_id) and data that could have an indirect impact on privacy and is ignored by *ReCon* (headers suitable for fingerprinting)

6.3 Evaluation Schema

Finally, the schema used for evaluation is described. Next to the selection of investigated applications and the respective basic usage patterns, also the considered privacy metric and the experiments are explained.

6.3.1 Example Applications

The evaluation should be performed using an iOS device for practical reasons.

App categories that should be investigated are weather apps and news apps. Reason for this selection is, that apps of these categories have a rather simple structure: they just provide content and don't rely on user generated content (which should reduce traffic on the upstream and thus simplify analysis). Also, some apps might incorporate advertisements which is beneficiary for the evaluation of the blockhost approach. Weather apps could be especially interesting, as they often rely on location data which could be obfuscated easily and without significant effects on the functionality.

Inside the chosen categories, the most popular free apps should be selected for the evaluation. As the evaluation of the apps is done manually, three apps for each category should be selected. To find suitable apps, a query to find the top iOS apps for Germany in both categories *News* and *Weather* were performed on the website Similarweb. In tables 6.4 and 6.5, the top five results can be found.
For weather-apps, the first three apps are chosen for analysis. In the *News* category, the first two ranks are occupied by *Twitter* and *reddit*. Both heavily rely on user-generated content and are not traditional news apps – *Twitter* is a microblogging website [24], *reddit* is described as "[...] an aggregator of user provided content [...]" ([3]). User-generated content is expected to change more quickly than content of traditional news apps. As this may reduce reproducibility in later data collection steps, these apps are not considered and instead places 3-5 of the ranking are chosen.

6.3.2 App Functionality & Simulated App Usage

As different apps may include different features (like push-up notifications or included video-streams), a basic usage pattern should be defined for each app. This pattern should then be utilized for both data generation and assessment of functionality retention. While each pattern is specific for one app, the patterns are similar in the categories. In the category *news*, the pattern focuses on reading textual articles. For apps from the *weather* category, the basic usage pattern consists of getting weather information for the current location and for another city (by searching for a name). In table 6.6 two examples (one per category) for defined usage patterns can be seen. Usage patterns for all assessed apps can be found in the appendix A.2.

Rank	Name	Developer
1	WetterOnline	WetterOnline
2	wetter.com	wetter.com GmbH
3	RegenRadar	WetterOnline
4	My Aurora Forecast & Alerts	JRustonApps B.V.
5	Echtes Thermometer	cong qi

Table 6.4: Top iOS app ranking by Similarweb for weather apps in Germany [52]

Rank	Name	Developer
1	Twitter	Twitter Inc.
2	Reddit	reddit
3	tagesschau	ARD Online
4	ntv Nachrichten	n-tv
5	ZDF-heute	ZDF

Table 6.5: Top iOS app ranking by Similarweb for news apps in Germany [53]

ZDFheute	**wetter.com**
Open App	Open App
Click 1st Article	Popup "Hyperlokales Wetter": Select No
Scroll to Bottom	Search for place: Berlin
Click 2nd Article	Scroll to Bottom
Scroll to Bottom	Open Menu; Enable "automatischer Ort"
Click "Ticker"; Don't Scroll	OS-Tray: "Allow Access to Location Once"
Open Tab "Stories"; Go through 1st Story	Scroll to Bottom
Open Tab "TV"; Play Main Video for 3s	

Table 6.6: Example of defined usage pattern for *ZDFheute* (category news) and *wetter.com* (category weather)

6.3.3 Measure Privacy Improvement

In a comparison of web privacy protection techniques, Mazel, Ganier and Fukuda evaluated different PETs and the metrics they used to measure privacy improvement [40]. Often, the number of HTTP requests, number of contacted domains, amount of received cookies or a privacy footprint[15] is used. However, the metrics using the amount of requests/cookies/domains only operate on a high level and not directly on data points. Similarly, the privacy footprint rather focuses on the *interconnectedness* of first and third parties ([36]) than on actually leaked data points. Therefore, a different metric based on granular data points was defined and used to measure the privacy improvement.

The core assumption for this work is that an increase of privacy is equal to less personal information shared with any external party. Therefore, the number of data points sent to service providers or a third parties is used as a metric. The data points are further divided into the four categories defined in chapter 6.1.3 to allow more fine-grained analysis.

As the privacy proxy should improve user's privacy by obfuscation, data points which contain blank data are considered to not have an impact on privacy. For this thesis, the numeric data *0* and character data which only contains *a*'s are considered as blank data and referred to as *anonymized* data. This is arbitrary to some extent, but facilitates later analysis.

[15] "The privacy footprint represents interactions between first parties and third parties (i.e. potential trackers) in a graph" ([40])

6.3.4 Experiments for Data Generation

The generation of data for later analysis should be done according to the following descriptions. Analysis of the caching approach focuses on other aspects, therefore a different procedure is used.

Privacy Proxy
For each of the three modes *no interference*, *blockhosts* and *anonymize*, the basic usage pattern for all six apps should be simulated. To retain full reproducibility, each app was reinstalled between each mode (so every basic usage pattern is performed on a fresh installation of the app). *mitmdump* is used to store the resulting flows.

Caching Approach
Generation of data for the caching approach looks a little different. The same set of operations was performed twice for all six apps (with mode *anonymize*). This allows the cache to be filled during the first run and to serve cached responses in the second run.
The apps were not reinstalled between the two data generation runs. But a dummy run was performed before the first run to counteract behavior only visible on the first start of an app (like accepting of cookies or other setup-related actions).

Besides the flows captured by *mitmdump*, also a look at the logs provided by *squid* should be taken. To guarantee a clean start, the cache is cleared and existing log files are deleted right before the first run.

7 Implementation

In this chapter, technical aspects of the privacy proxy implementation are described. First, an overview over existing software which is suitable as a basis for the proxy is given. For the caching approach, different strategies and associated software are presented. Afterwards, setup details for the smartphone and the proxy are explained. The chapter closes with a description of the developed add-ons that are integrated with the existing tools and actually increase the privacy.

7.1 Overview over Suitable Software

First, existing software used for the man-in-the-middle attack is outlined. Afterwards, different caching strategies and associated software solutions are presented.

7.1.1 mitmproxy

mitmproxy is a proxy software which allows to intercept (encrypted) HTTP[16] traffic. When a flow is intercepted, all kinds of actions can be performed: modification of request/response data, blocking of requests/responses or forwarding of the unchanged communication.

mitmproxy contains three front-ends for the core proxy functionality:

- **mitmproxy console** is a console interface that allows to interactively intercept HTTP traffic
- **mitmweb** offers a web-based GUI
- **mitmdump** allows to programmatically modify traffic and is able to write/read flows to/from files. Unlike the previous tools, it doesn't keep the traffic in memory

While *mitmweb* is used mostly during the development of the privacy proxy, *mitmdump* should be used in later production setups for performance reasons. All these tools provide extensive add-on support. The implementation of the privacy proxy will also be done using the add-on architecture. For the evaluation, *mitmdump* is used to persist the data of generated flows. These stored flows will then be analyzed regarding the privacy increase.

[16] *mitmproxy* supports *HTTP/1*, *HTTP/2*, and *WebSockets* protocols [42]

7.1.2 Caching Software

Selection of suitable caching software was more complex, as different strategies were tried out. The main reason for the experimentation with different software was to come up with a solution that requires only a single interception step at the proxy – decryption and re-encryption of traffic should only happen once. This was desired for performance reasons. However, the following two approaches trying to do so failed. Thus, *mitmproxy* was configured to forward its traffic to the caching software *squid*. In this setup, traffic is decrypted and re-encrypted twice (at *mitmproxy* and at *squid*).

Strip Encryption with mitmproxy

One potential solution with a single interception step is to strip the encryption with *mitmproxy* and then forward the decrypted traffic to some caching software. If the caching process requires to fetch content from the original target server, an encrypted connection to it should be established.

It was possible to make *mitmproxy* forward plain HTTP instead of HTTPS (and thus "downgrade" HTTPS) with a simple add-on. The functionality was verified by successfully requesting websites which deliver content over both HTTP and HTTPS. However, the combination with *squid* caused issues when requesting HTTPS websites via this method. *squid* replies with `502 Bad Gateway` and closes the connection[17].

Since the combination with *squid* did not work, other solutions were assessed.

Parrot Cache

One investigated idea was to use caches as "parrot" caches. This means that (unlike regular cache mechanics) the caching software can't access the requested servers itself, but actively gets the content (that it should replay) pushed onto. In the envisioned scenario, *mitmproxy* would check whether the requested data is cached. If it is, the data is replied to the client. If not, *mitmproxy* would forward the request and then "push" the response on the cache. In this case, again only a single interception step is required.

Different tools were assessed to employ such a parrot cache: *Varnish* and *Apache trafficserver*.

Varnish is originally designed as an HTTP reverse proxy and it is usually put in front of webservers to increase website performance. An additional feature are VMODs: extensions written for the *Varnish* cache. These extensions allow defining policies how *Varnish* should handle incoming requests. Therefore, an existing "parrot" VMOD was rewritten to match the current *Varnish* version[18].

This solution works, but is limited to specific file sizes. The VMOD framework only allows to access request headers[19]. Therefore, the content, that should be cached, must be transformed into a header field. As the targeted apps regularly rely on multimedia assets which would exceed the maximum length of header fields, this approach is considered impractical.

The next assessed caching software is *Apache trafficserver*. In the documentation, a way to push data on the cache is described[20]. However, it wasn't possible to push a simple object

[17] When checking the *squid* logs, a `CONNECT` request to port 443 can be seen. This should not be there, as the scheme was set to `http` and the port to `80`. This might be a bug in the *mitmproxy* software

[18] Original extension: `https://gist.github.com/theinel/ab2014eef9765650764b`.
Adopted VMOD: `https://gist.github.com/frid000/b94b92b29eb78bbf3641ca529bc131dd`

[19] The enterprise version of *Varnish* allows to also access the request body. But an email asking for access for academic reasons remained unanswered

[20] `https://docs.trafficserver.apache.org/en/7.1.x/admin-guide/storage/index.en.html#pushing-an-object-into-the-cache`

into the cache – even though the instructions were followed exactly. *trafficserver* always returned `400 Malformed Pushed Response Header` instead of writing the object to the cache. After some unsuccessful experimentation, this solution also was considered infeasible.

Squid

As the approaches employing a single interception step failed, the *squid* caching software was set up. It implements a feature to pose a man-in-the-middle attack on traffic: `sslBumping`. With it, the final flow of traffic between the client and the requested server looks like this: *Client* → *mitmproxy* → *squid* → *Targetserver*. At the points *mitmproxy* and *squid*, the encryption is stripped and reestablished which means that there is one repeated decryption/encryption step in the whole process. As the target of this work is only to prove the concept, potential performance issues were acceptable.

One potential solution to this solve this issue in the future could be to transfer the privacy increasing mechanisms performed at *mitmproxy* to an Internet Content Adaptation Protocol (ICAP) service[21]. ICAP services can perform transformations or other processing on HTTP messages [31]. With this setup, only one encryption/decryption step would be performed. However, the transition from *mitmproxy* to an ICAP service exceeds the timing for this thesis.

Squid was configured to include options to cache as much as possible and ignore cache control fields sent with requests[22]. The ignorance of such fields violates the HTTP standard, but is accepted for the evaluation in this thesis [54].

7.2 Setup of Hardware

This short section describes details of setting up the Raspberry and the smartphone.

Setup of Raspberry

Basis for the proxy is a Raspberry Pi Model 2B with a WiFi dongle. Its setup is rather straight-forward. After setting up the WiFi access point with *hostadp*, simply the *iptable* routing has to be setup correctly to forward all traffic coming from the WiFi interface to *mitmproxy* and then to the interface providing internet access.

Setup of Smartphone

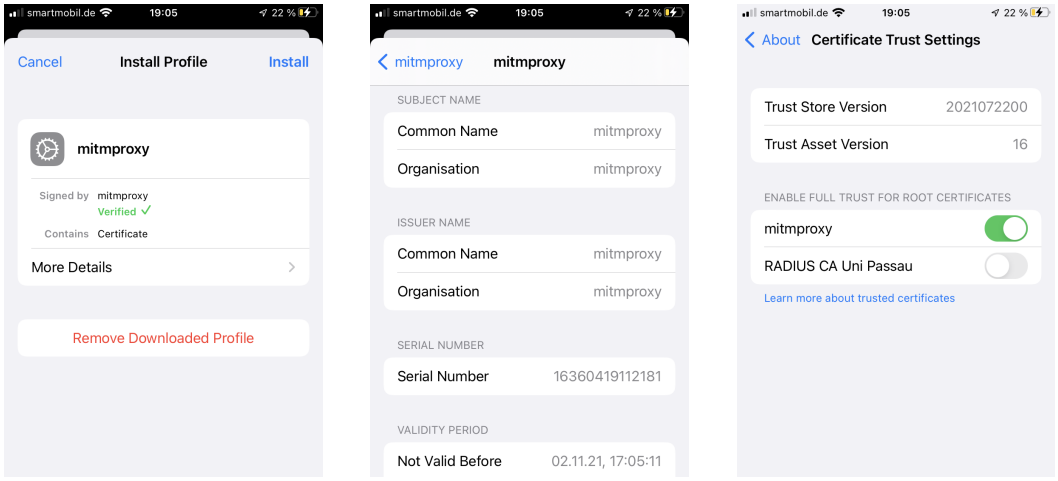
On the smartphone, only the root certificate generated by *mitmproxy*'s CA has to be installed. On iOS devices, this is a rather simple task: the certificate has to be installed and afterwards a setting to enable full trust for root certificates has to be enabled. Figure 7.1 illustrates the steps.

7.3 Developed mitmproxy Add-ons

In this section, the developed add-ons for *mitmproxy* are described. The add-ons could be divided in two groups: the ones that modify traffic in real-time ("blockhost", "anonymize") and the ones that work on "static" traffic flows previously dumped by *mitmdump* ("detect_breaches", tools for categorization of data points).

[21] `https://wiki.squid-cache.org/Features/ICAP`

[22] Passed options to the refresh pattern: *override-expire override-lastmod ignore-reload ignore-no-store ignore-private store-stale*



(a) Prompt to install certificate (b) Certificate details (c) Setting to trust root certificate

Figure 7.1: Process of installing a root certificate on an iPhone

To illustrate the structure of the add-ons, an example add-on is depicted in figure 7.2. It shows an add-on that defines the most important event hooks that will be called by *mitmproxy*. Methods `__init__` and `done` are executed at the time *mitmproxy* loads/closes the add-on. In this part, set-up functionality like loading files or closing actions like writing data to the disk could be performed.
The methods `request` respective `response` are called whenever a request/response is intercepted at the *mitmproxy*. Then, these methods could analyze or modify the HTTP message – in this case the header "request_counter" is added to all requests. Furthermore, the shown add-on checks every response whether it contains an image. If so, a counter variable is increased.
When the `request` or `response` method is finished, *mitmproxy* forwards the HTTP message to the intended recipient.

```
class ExampleAddon:
    def __init__(self):
         self.num = 0
     self.img_cnt = 0
    def done(self):
        with open(logfile, "w") as f:
            f.write("max num: ",  self.num)
    def request(self, flow: HTTPFlow):
        req = flow.request
        req.headers["request_counter"] = self.num
        self.num = self.num + 1
    def response(self, flow: HTTPFlow):
        res = flow.response
        if (res.headers["content-type"][:5] == "image"):
            self.img_cnt = self.img.cnt + 1
addons = [
    ExampleAddon()
]
```

Figure 7.2: Example add-on for *mitmproxy*

```
{
  "o_first_lvl" : "val_1",
  "a_first_lvl" : [
    "a_val_0",
    "a_val_1"
  ],
  "nested_o_first_level": {
    "o_second_lvl" : "val_4"
  }
}
```

(a) Example JSON object

Key (or "path")	**Value**
o_first_lvl	val_1
a_first_lvl/0	a_val_0
a_first_lvl/1	a_val_1
nested_o_first_level/o_second_lvl	val_4

(b) Key-value representation of the JSON object

Figure 7.3: Visualization of conversion of a JSON object into a set of key-value pairs

7.3.1 "Static" Add-ons

First, the add-ons which work on flows recorded by *mitmdump* are presented. Some helpful tools for the categorization of data points according to their impact on privacy are briefly outlined. Then, the add-on which traverses recorded flows and summarizes the amount of observed data points is described.

Tools for Categorization

The first developed add-ons yield the basis for the categorization of data points according their impact on privacy. Basically, the tools traverse the given recorded flows and extract every key for each header or URL parameter and an example value.

The body-part is more complex, as it could contain data in different formats. Besides the `url form` and `byte form` format (which again consist of key-value pairs), JSON-formatted data was often observed. Therefore, it was decided that the privacy proxy should also support the JSON data format.
JSON data consists of nested key-value pairs and arrays. To be able to reuse the existing architecture, it is required to categorize single data point in such a nested data structure. Therefore, it was decided to firstly convert the JSON data into a set of key-value pairs. This is performed by defining the final key of a data point as the combination of all keys that lead to the point (the combined keys can also be seen as a "path" through the JSON structure). The value is the same in the old and the new representation.

To clarify the procedure, an example is given in figure 7.3. On the left, an exemplary JSON object (7.3a) is depicted. The corresponding key-value representation can be seen on the right (7.3b). In the shown example, the array index is used as a part of the path. For my categorization, the array index will be replaced with a placeholder to avoid categorizing the same path multiple times and to handle arrays of varying lengths.

As it was observed that one service encodes its JSON data with base64 before transmission, the ability to decode base64 for requests by this service was implemented.

The result of these tools are all observed keys (in headers, URL parameters and bodies) in the traffic and corresponding example values. Based on this information, the categorization as explained in chapter 6.1.3 is performed.

Detect Privacy Breaches

Then, an add-on to evaluate the proposed privacy proxy is developed. It again traverses the headers/bodies/URL parameters of all HTTP requests in stored flows and then checks each

found key-value pair for two aspects:

1. Whether the found value is anonymized (meaning the value only contains *a*'s or 0)
2. The category of privacy impact of the respective key

Here should be noted, that values considered as anonymized does not necessarily mean that the privacy proxy replaced the value! It could also occur that the data point contains *a*'s or zeroes by nature. This drawback of the metric should result in the appearance of anonymized data in operation modes *no interference* and *blockhost* – even though the anonymizing mechanism is not used in these modes. This issue adds some noise to the results, but that noise is expected to be low.
In return, this method offers the advantage that the anonymizing add-on and the evaluation add-on are separated. Hence, the evaluation is performed on the traffic that was actually transmitted. This allows the evaluation to be as close to the reality as possible.

When the category and the anonymized-status is determined, a respective counter variable will be increased. The dimensions of the variables can be seen in figure 7.4.

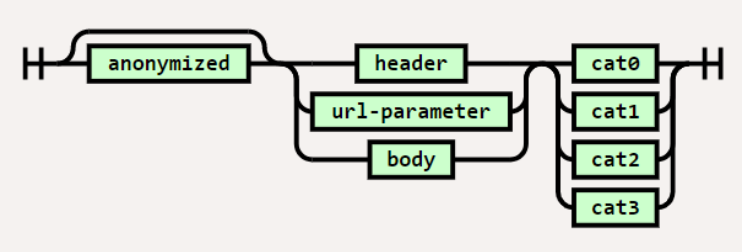


Figure 7.4: Dimensions covered by variables: are data points anonymized, their location and the category of privacy impact they pose

One threat to privacy are tracking pixels. As they are hard to recognize in traffic (the requests could also just try to obtain image files, see chapter 6.1.2), a separate mechanism to detect them was introduced. It checks each HTTP response for it's content type. If the content-type is *image*, the image is parsed. Then, the dimensions are extracted and if the dimensions are $\leq 1px * 1px$, this response is counted as an occurrence of a tracking pixel.

For the analysis of the caching approach, another dimension was added to the already considered counter variables: how the cache reacted to a request. This data was read from the `Cache-Status` header of each response. It contains additional information how the cache handled the request: with a hit or a miss and the reason for it [32].

To conclude, the add-on which detects privacy breaches traverses all data points in dumped traffic. It counts the number of observed data points for each category of privacy impact, the place where it was located and whether it contained anonymized data or not. These summaries can then be used for later data analysis.

7.3.2 "Realtime" Add-ons

Next, the add-ons that interact with traffic in realtime are described. The first add-on takes care of blocking known T&A servers for the *blockhost* mode of the privacy proxy. The second add-on contains the obfuscation of data points as specified by the *anonymize* mode.

Blockhost

The first add-on which interferes with traffic passing the privacy proxy, is the mechanism to block known T&A servers. While the underlying mechanism is trivial – just compare the

target of the request with a predefined list of hostnames and block the request if a match occurred – the selection of said list is not.
There are plenty of hostname lists, each with a slightly different objective and different included hostnames[23]. As the list easily could be swapped and adapted to personal requirements/preferences, it was decided to use the standard list which is shipped with *pi-hole*: *Steven Black's host list* [10].

Anonymize

Developed next is the add-on which actually obfuscates (by replacing/blurring) data seen in traffic. As previously stated, three parts of a request should be investigated and modified if necessary: headers, URL parameters and the body. The first two parts headers and URL parameters are easy to analyze, as both already contain data in key-value format. For the body part, JSON objects are first transformed into key-value pairs (as seen in chapter 7.3.2).

When a request arrives at the privacy proxy, the *anonymize* add-on checks each key-value pair contained in that request. For each key, it checks the corresponding category of privacy impact. Based on the category, the method of obfuscation is selected:

- **Category 0** (technical data points): no obfuscation
- **Category 1 & 2** (fingerprinting data, unique identifiers): replacement of values
  - Data points of type `string` are replaced with concatenated *a*'s of the original length
  - Numeric data points are replaced with zero
- **Category 3** (geolocation data): blurring by rounding the GPS coordinates to the first decimal place. This yields an accuracy of approximately 11km [23].

[23] `https://github.com/topics/pihole-blocklists`

8 Evaluation and Discussion

In this section of the work, the results for the evaluation of the developed privacy proxy are described. Afterwards, the findings are discussed in their context.

8.1 Results

This chapter addresses the results of the performed experiments with the different modes of the privacy proxy (*no interference*, *blockhosts* and *anonymize*). First, a glimpse on issues encountered during the experiments and the quality of the collected data is taken. As the developed privacy proxy broke some functionality of apps, some adjustments to regain app functionality had to be performed. These adjustments are described next. Then, the gain of privacy provided by the privacy proxy and the caching approach is shown. Finally, observed obfuscation methods employed by service providers are presented.

8.1.1 Issues Experienced During Data Generation

While generating first test data for the implementation, the question of reproducibility of experiments rose. It is extremely hard to perform reproducible experiments when operating "in the wild" – meaning with real phones, real apps and real service providers.

One issue is that devices send a lot of background traffic. Updating of mails/weather/stock data, checking account settings and internet connection tests are all examples of traffic that could occur at any time. It is impossible to get rid of all background noise, especially the requests made from the operating system itself. One way to reduce the noise was to use a freshly reset operating system without any third-party apps installed (exempt the apps under investigation) and to only set up the required Apple account. Further, the option to use the WiFi network in low data mode was enabled. This feature reduces background data activity of third-party apps and built-in apps (it prevents for example automatic downloads/backups, stops article prefetching, turns off background app refresh, ...)[7].

Another impediment is the lack of insight in inner workings of apps and service providers. One feature that could reduce the reproducibility of experiments is the ability of apps to prioritize HTTP requests differently[24]. Such a mechanism could interfere with the observed traffic, as outside effects (like varying internet speeds or server loads) are impossible to avoid. In the worst case, a request with a low priority could be left out in one experiment

[24] `https://developer.apple.com/documentation/foundation/urlsessiontask/1410569-priority`

and served in another. Another problem is posed by conditional app behaviors. When a request fails (e.g. because it is blocked by the *blockhost* add-on), the app may try to resubmit it multiple times which may impact subsequent requests. Or it could report the occurred error to another server and thus influence the observed traffic.

These issues show that it is impossible to generate completely clean data and some noise in the resulting numbers is unavoidable.

8.1.2 Data Quality

To assess whether the proposed privacy proxy sufficiently covers the traffic, the share of measures that could be processed by the proxy was examined. In particular, every time the body of a request contained a data format which is not supported by the privacy proxy (e.g. *protobuf*) or the content could not be parsed, this was recorded. The same applies for key-value pairs that were observed in traffic but whose keys were not present in the privacy classification.

As can be seen in table 8.1, $\sim$ 99% of all keys seen in the captured data were categorized according their privacy impact across all modes of operation. The share of missed keys was slightly increased in the mode *anonymize*, but still very low.
One irritating number are the 198 data points whose keys were missing in the privacy categorization in the mode *no interference* – all keys observed in this mode were the basis for the categorization. Therefore, no keys should be missing in the privacy categorization here. But due to a bug in the helper tool for the classification, some keys were not categorized. As the majority of the keys still was covered and the regeneration of data for all modes is very time-consuming, regeneration was considered disproportional and the existent data is further used.

A similar check was done to support the design decision to only support the body formats URL-forms and JSON – and ignore further formats. Again, the majority of requests could correctly be processed. The results in table 8.2 show, that the share of requests the privacy proxy was unable to process ranges between 0.91% and 1.68%.

These results show that the supported data formats and the manual classification of keys cover the vast majority of observed data points and only a small percentage of traffic is ignored by the proposed solution. Thus, meaningful analysis could be performed on this data basis.

	No Interference		**Blockhosts**		**Anonymize**	
	Amount	Percent	Amount	Percent	Amount	Percent
Categorized Keys	27.696	99.29%	15.220	99.29%	15.376	99.02%
Missed Keys	198	0.71%	109	0.71%	152	0.98%
$\sum$	27.894	100%	15.329	100%	15.528	100%

Table 8.1: Amounts and percentages of data points whose keys were categorized according their privacy impact during evaluation

	No Interference		**Blockhosts**		**Anonymize**	
	Amount	Percent	Amount	Percent	Amount	Percent
Supported Formats	1.466	98.32%	1.412	99.09%	1.377	98.71%
Unknown Formats	25	1.68%	13	0.91%	18	1.29%
$\sum$	1.491	100%	1.425	100%	1.395	100%

Table 8.2: Amount and percentages of requests whose data format could be processed/not processed

Part of Request	Key	Old Privacy Category	New Privacy Category
Path Parameter	templates	1	0
Header	Api-Auth	1	0
Header	x-goog-api-key	1	0
Header	authorization	1	0
Path Parameter	geoobjectkey	2	0
Body	app-id	2	0

Table 8.3: Modifications made to the initial classification to retain app functionality

8.1.3 Retention of Functionality

One of the objectives of this thesis was to retain the functionality used in the basic usage patterns (defined in section 6.3.2) of the investigated apps. In the evaluation of modes *no interference* and *blockhosts*, no constraints regarding the functionality was observed.

Unfortunately, this was not the case for mode *anonymize* where data was obfuscated. In this mode, different functionality failed and the categorization of privacy categories had to slightly be modified for some keys to reestablish full functionality.

One failing feature of the basic usage pattern was video playback in the *ZDF* app (see table 6.6). After investigation, it appeared that the *Api-Auth* (intially considered as privacy category 1) header must contain valid data and can't be obfuscated. To solve this issue, the categorization of the respective header was switched to category *technical*. As a result, this specific header was not obfuscated anymore and the video replay worked again.

Another issue was a third-party cookie consensus platform employed by *wetter.com*. Without the appropriate adjustment it was simply not possible to accept or decline the cookie popup and therefore the app could not be used. But again, just a minor change of the path parameter *templates* fixed the issue.

While the solution to regain functionality was rather easy to figure out for these two apps – the HTTP responses contained useful error messages – this wasn't the case for all apps. *WetterOnline* for example required larger adjustments. This was owed to the fact that they apparently rely on a Google service to acquire an `api-key` which then is used to authorize at the target servers. Additionally, the initial categorization lead to the obfuscation of the field *geoobjectkey*, which is used to identify cities on the backend. As this information is crucial to get the weather data for the correct city, the privacy categorization for this key was also changed. All such modifications, that were necessary to regain functionality for all apps can be found in table 8.3.

While it is true that the modification of the initial categorization enables leaks of potentially privacy impacting data, the retention of functionality should have a higher weight from user's perspective. Also, it is worth to note that only very few changes had to be performed and most of the changes were done on authentication related keys. The impact on user's privacy therefore is expected to be limited.

8.1.4 Improvement of Privacy

In this chapter, the effects of the modes *blockhost* and *anonymize* on the user's privacy are presented. First, a look on tracking pixels is taken. Then the effects on the number of leaked data points are depicted.

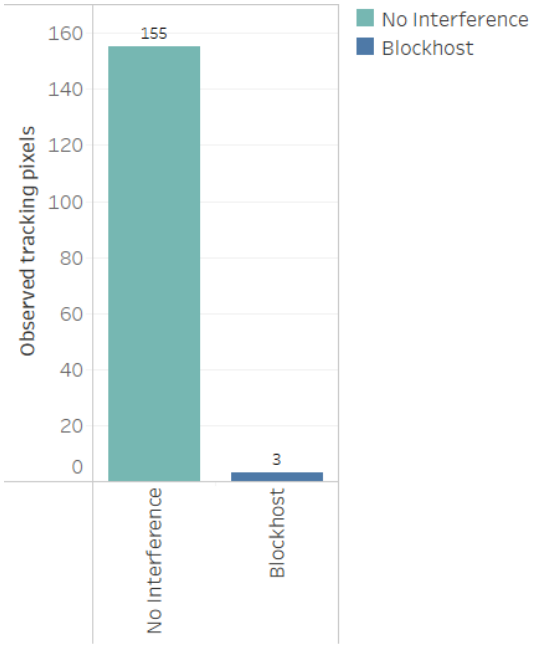


Figure 8.1: Comparison of occurrences of tracking pixels in operation modes *no interference* and *blockhosts*

Reduction of Tracking Pixels

Figure 8.1 shows the occurrences of tracking pixels in the operation mode *no interference* and the mode *blockhost*. As can be seen, the number of tracking pixels that have been observed in traffic are drastically reduced. This shows that the well-known method of blocking known T&A hosts indeed yields an increase of privacy. Nevertheless, a few tracking pixels still were observed. This may be a hint that the selected list of known hostnames may lack some entries.

Effect on Number of Leaked Data Points

In this section, the effects of the proposed privacy proxy and the different operation modes on the number of leaked data points will be assessed. For this analysis, primary figure 8.2 will be used. It shows the number of observed data points in the flows from the experiments. Each of the three assessed modes is depicted: *no interference*, *blockhost*, *anonymize*. Each of the modes is further divided into two columns: *non-anonymized* data and *anonymized* data. Recall here, that data is considered as anonymized when it exists purely out of zeroes or *a*'s. For each of the both "states" of data content, the four previously defined categories[25] are shown.

As the chart is rather complicated to read, a reading example will be provided. A look at the second bar from the left in the mode *no interference* will be taken (10.359 data points). This bar represents 10.359 data points which are classified as category 1 and contain data which is not considered anonymized. They might appear in the headers, path or bodies of HTTP requests and were recorded while performing the defined basic usage pattern for all six apps.

At first, a look on the difference between the baseline (mode *no interference*) and the blocking of known T&A hosts will be taken. A decrease can be seen on the amount of data points

[25]Quick recap: 0: technical; 1: fingerprinting or similar; 2: unique identifiers; 3: geolocation data

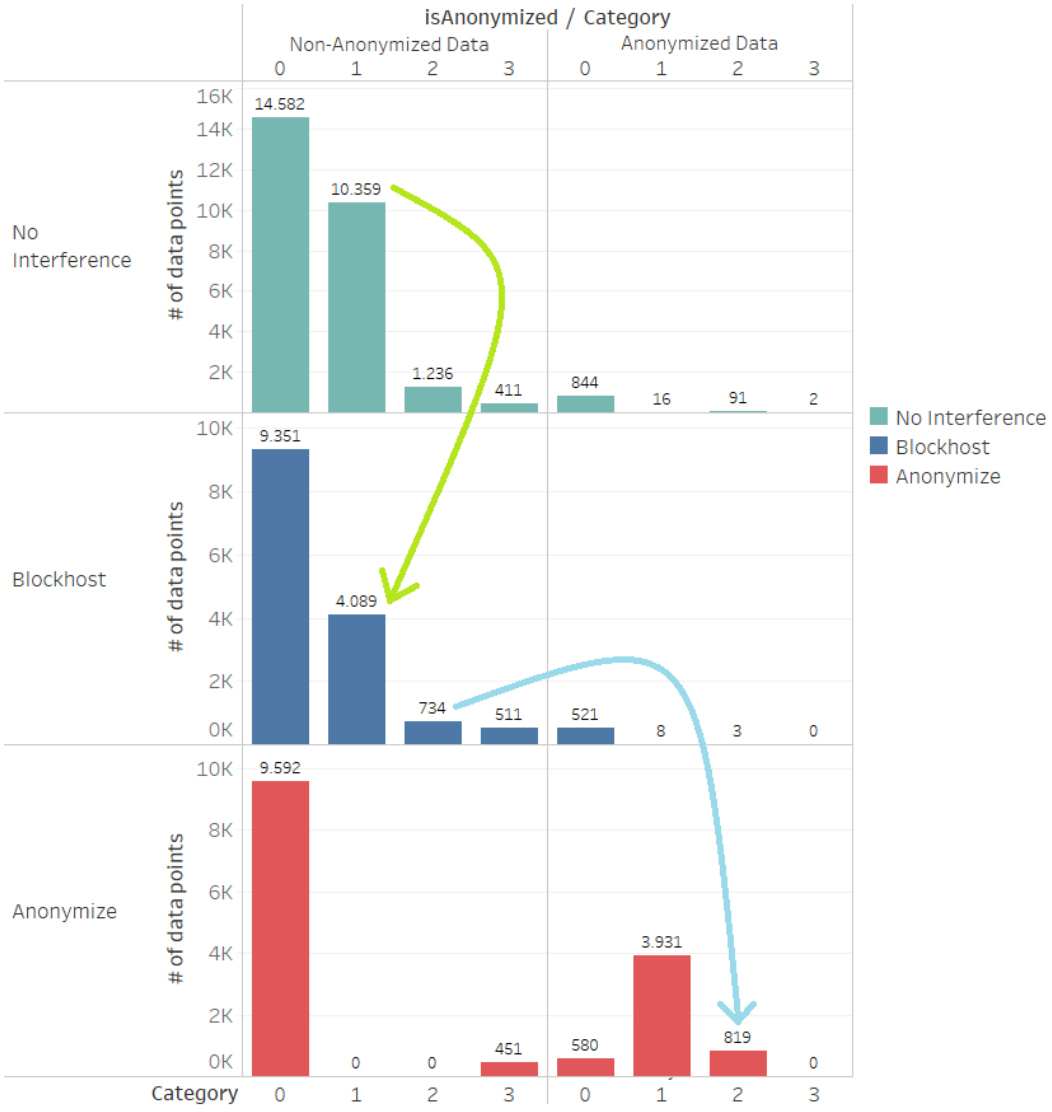


Figure 8.2: Absolute number of non-/anonymized data points observed per mode and category. Improvement of privacy is exemplary shown for two categories with arrows. The green arrow shows the reduction of seen data points of category 1 by blocking known T&A hosts. The arrow in color cyan shows the shift of category 2 data when applying the *anonymize* mode

```
GET https://api-app.wetteronline.de/app/weather/water?altitude=550&av=1&latitude=48.12&
longitude=11.52&mv=10&timezone=Europe/Berlin HTTP/2.0
```

(a) Path parameters of request in mode *no interference*

```
GET https://api-app.wetteronline.de/app/weather/water?altitude=000&av=1&
latitude=48.1&longitude=11.5&mv=10&timezone=Europe/Berlin HTTP/2.0
```

(b) Path parameters of request in mode *anonymize*

Figure 8.3: Comparison of two requests in modes *no interference* and *anonymize*. Geolocation data has lower resolution in the second request

considered under categories 0, 1 and 2. Non-anonymized data from category 0 drops from 14.582 data points in the mode *no interference* to 9.351 data points, category 1 dropped even more: from 10.359 to 4.089 data points (also visualized by the green arrow). This indicates that a good portion of the traffic which was present in mode *no interference* was targeted to known T&A hosts and thus was blocked in the *blockhost* mode.
When inspecting category 3 (geolocation data) in the *no interference* mode, a decrease can only be seen in the anonymized data (from 2 data points to 0). Observed non-anonymized geolocation data points occurred even more often (from 411 to 511). This is an interesting finding, as geolocation data apparently is not transmitted to T&A hosts at all (which is why there is no decrease of observed data points in non-anonymized data) or only with dummy values zero (which explains the reduction of anonymized data points). The increase of observed non-anonymized data points could only be explained by noise in the generated data.
Another potentially counterintuitive aspect is the presence of anonymized data points in the *no interference* mode: in this mode no obfuscation is performed. However, this can easily be explained by the underlying mechanism used to detect anonymized data: checking whether the value of data points consists only of zeroes and *a*'s. The data points visible in the anonymized section are values considered as anonymized by nature.

Next, the differences between the modes *blockhosts* and *anonymize* will be inspected. Here, the total amount of data points seen in the experiments is similar – they are just distributed differently in the *isAnonymized* dimension. The shift (exemplary pointed out by the cyan arrow for data points of category 2) is the effect of the implemented obfuscation mechanism that replaces data points of category 1 & 2 with anonymized dummy values.
One aspect that needs explanation is the difference of values observed in the *blockhost* and the *anonymize* mode. When looking at the exemplary highlighted shift of category 2 data (cyan arrow), the number of observed data points increased from 734 to 819 data points. In a completely reproducible experiment, both numbers should be the same. However, the data generation was performed "in the wild" which inevitably introduces noise and thus influences the here shown numbers.
Again, data points containing geolocation data (category 3) were not affected – they stayed at the *non-anonymized data* column. Yet, they were blurred during the anonymization step. The reason for the missing shift to the anonymized column is, that the implementation is unable to detect this blurring step – it is hard to detect whether an observed float number is rounded or just sent with the specific resolution. Manual inspection however proved that the blurring indeed happened (see figure 8.3) and the precision of the coordinates was reduced.

To summarize, it was shown that the number of data points observed in traffic could indeed be decreased by just blocking access to known T&A hosts. In the next step, the feasibility to replace a majority of said data points with "anonymized" data while maintaining functionality was shown.

Code	Explanation
TCP_MISS	The requested object is not in cache
TCP_HIT	A copy of the requested object is in cache
TCP_MEM_HIT	A copy of the requested object is in cache and is stored in RAM

Table 8.4: Excerpt of result codes used in *squid* logs

8.1.5 Caching Approach

While evaluating the caching approach, it occurred that the *squid* caching software regularly crashed. After deeper investigation, the underlying issue[26] was found. The bug was fixed quickly by the developers, however only in the development release and not the stable release recommended for production usage. Therefore, the following evaluation was performed on the development release of *squid*[27].

To evaluate whether it is possible to use caching with apps and the resulting effect on privacy, data of two sources was inspected: *squid*'s access logs and dumped flows of the experiments performed for the caching approach. The log files contain *squid* specific result codes 8.4. The most important ones are explained in table [17].

Analysis of Log Files

Figure 8.4 shows the results of the log file analysis. The red reference line divides the first run from the second (which is shaded). The upper row describes the number of requests that were missed by the cache, the lower refers to the requests whose responses were present in the cache. It should be noted, that in the figure only memory-hits are displayed. This is due to the reason, that there occurred only a handful of hits on content stored on the disk. For completion, the respective graph can be found in the appendix A.3.

As can be seen in figure 8.4, the number of cache misses was clearly higher in the first run and then declined during the second run where the cache was filled with some content. From the lower row, which is displaying the number of cache hits, one can see that the first thirty seconds are without any hit. This is owed to the empty cache, which needs to be filled before yielding any hits. In the second run, more frequent hits can be seen.
One exception is the spike of hits on second ~ 159 in the first run. As this spike is unexpected, a closer look was taken. As it turned out, the spike consists of a single request that was repeated 18 times. The first time, the cache stored the response and the following requests were answered with the stored response. Repeating the same request multiple times shows the opaque behavior of applications.

To summarize, it can be said that the cache works as intended. It starts with a lot of misses while the cache is filled. In the second run the cache hits increase. However, it should be noted that still parts of the traffic weren't caught by the cache. This fact should be further investigated in consecutive work.

Analysis of Effect on Privacy

In this section, the effect of the caching approach on privacy is evaluated. For that, a look is taken at the data points that were observed during the second run of the experiment performed for the caching approach. The second run is considered, as in this case the cache is already filled and thus has the ability to increase privacy. In figure 8.5, the results are seen – distinguished on the caching status. What stands out first, is the small number of

[26] `bugs.squid-cache.org/show_bug.cgi?id=5090`
[27] specifically squid-6.0.0-20220115

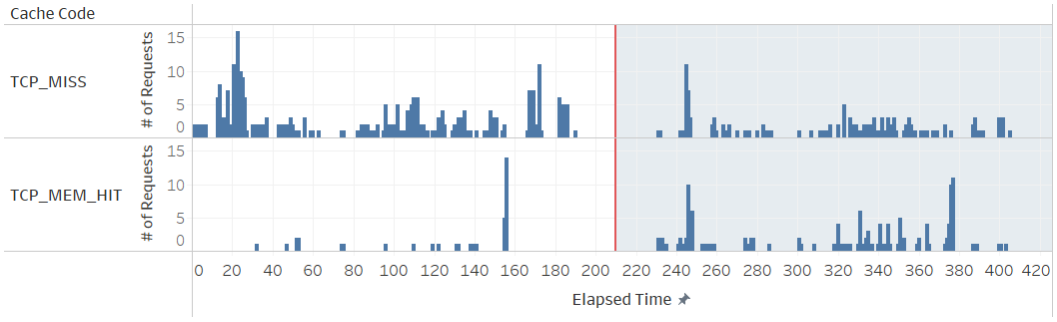
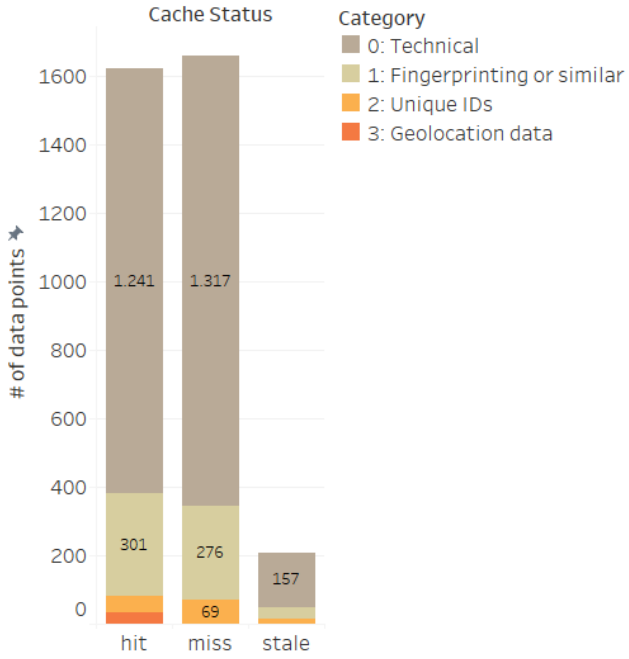


Figure 8.4: Analysis of cache misses and hits in mode *anonymize*. The red reference line splits the first (left) and the second (right, shaded) run

overall observed data points in the caching experiment. Compared to the analysis made earlier, only a fraction of data points is seen. This is owed to the fact, that the experiment performed for caching differs from the earlier performed data generation. In previous data generation steps, the apps were started for the first time while for the caching approach the apps already have been launched before. This reduces the observed data by data that is just transferred at the first launch of an app – like configuration files that are downloaded. Yet, the results should be resilient, as the two data generation steps in the caching experiment base on the same performed actions. Also, the total number of observed data points between the two runs is similar (run 1: 3.679 data points, run 2: 3.490 data points).



(a) Visualization of the data

	Hit	**Miss**	**Stale**
Cat. 0	1.241	1.317	157
Cat. 1	301	276	33
Cat. 2	47	69	16
Cat. 3	33	0	0
$\sum$	1.622	1.662	206

(b) Absolute numbers of data points observed

Figure 8.5: Number of observed data points during the second data generation run. Differentiated on the cache status and the category of the data points

In figure 8.5a, the data points are shown for the second run. As can be seen, a little less than half of all data points were part of requests that were cached. The distribution between the privacy categories is also half-and-half for categories 0, 1 and 2. Especially interesting

Method	POST	
URL	https://de.ioam.de/tx.io	
Headers	Content-Type	application/x-www-form-urlencoded
	Accept	*/*
	User-Agent	TagesschauNew/2911 CFNetwork/1329 Darwin/21.3.0
Body	$ae = eyJwcm90b2NvbFZlcnNpb24iOjEsInN0YXRzIjp7IklPTENvbmZ...$	
Body (decoded)	$ae = \{"protocolVersion" : 1, "stats" : \{"IOLConf...$	

Figure 8.6: Example excerpt for a request which employs some form of third party obfuscation: URL form data containing a single entry with a base64 encoded JSON object

is the fact, that data points which contain geolocation data all could be served with cached data and this information therefore wasn't leaked to any service provider or third party in the second run.
Also noticeable is the amount of stale data. This data could potentially be reduced by further fine-tuning settings of *squid*.

These results show, that caching indeed could increase privacy. However, still half of all observed data points were forwarded to service providers or third parties. Here, subsequent work should further investigate how to increase the amount of cached responses.

8.1.6 Observed Third Party Obfuscation

As service providers or T&A companies have interest in the transmitted data, they could employ mechanisms to make analysis and interference of request content harder or prevent it at all. Such methods could be additional encryption or obfuscation through e.g. hash functions [50]. To differentiate obfuscation performed by service providers or third parties from the obfuscation mechanisms employed in this thesis, this obfuscation will be referred to as "third party obfuscation".

One party which employs some sort of third party obfuscation is the digital audience measurement provider *Infoonline*. The mechanism itself is rather to nest formats than to avoid access to the request content, but it still is worth to mention. In figure 8.6 such an example request is depicted. As can be seen, the *content-type* in the header refers to a *application/x-www-form-urlencoded* form[28]. In that form, a single entry can be found: *ae*. Its value is actually a base64-encoded JSON object. As requests with this structure were observed frequently, the privacy proxy was extended to also decode base64 encodings.

A similar behavior is seen in some requests to google-analytics. In figure 8.7 such a request is depicted. As can be seen, the header *content-type* again indicates *application/x-www-form-urlencoded* payload. However, the actual content of the body is a JSON object. This contradiction could be handled programmatically, but this corner case occurs very rarely – across all 4.311 requests observed in the three modes of the privacy proxy, only 7 times this odd behavior was seen. Therefore, it was chosen to ignore this case.

The only further third party obfuscation method, that was observed are uncommon body formats. One request, for example, had content-type url-form, but the body contained mixed string and hex data like `\x08 \x04\_pfo\x18\x00 \x02\_o\x12\x04auto \x06 \x02` . Other requests directly send data with the content-type octet-stream. These data was also tried to manually decompress/parse with respective python libraries, but without any success. While those finding negatively impact the soundness of the system, only a small fraction

[28]The *application/x-www-form-urlencoded* format provides a way to encode name-value pairs [58]. It refers to the body of an HTTP request and not to the URL as one might think

Method	POST	
URL	https://www.google-analytics.com/mp/collect?firebase_app_id=1:263533392...	
Headers	accept	*/*
	content-type	application/x-www-form-urlencoded
	user-agent	wetter.com/748 CFNetwork/1329 Darwin/21.3.0
	...	
Body	{"*events*" : [{"*name*" : "*screenview*", ...], "*app_instance_id*" : "*a12a0a691ae3*"}}	

Figure 8.7: Example excerpt for a request which has content-type *x-www-form-urlencoded*, but actually contains JSON in the body. Further data is transmitted via URL parameters

of the observed requests actually fall under this category. This can be verified by table 8.2 in the data quality section (8.1.2), which shows that the vast majority of traffic could be parsed and processed by the privacy proxy.

In the end, it should be noted that nesting of formats also could originate from technical requirements and must not result from third party obfuscation methods.

8.2 Discussion

In this section, the previously presented findings are discussed and put in context. First, the impact of the privacy proxy on the smartphone user's privacy is investigated. Then, a look at the caching approach is taken. Finally, the observed third party obfuscation is discussed.

8.2.1 Improvement of Privacy

The previous results showed that a lot of (potentially privacy impacting) data is transferred to service providers and third parties. The blocking of known T&A hosts already harshly reduced the number of leaked data points. However, still a lot of data is transferred. This data however could be obfuscated in the *anonymize* mode and thus additionally increase the privacy.

One unexpected result was how few entries of the initial categorization had to be changed in order to regain functionality. This finding also indicates, that a lot of the transmitted data is not absolutely required to deliver content, but is additionally collected for tracking, analysis or other reasons. But it must be acknowledged, that adjustments to the initial categorization had to be made to regain app functionality. Thus, some data which might impact privacy may leak.

Also, it should be noted that this work relies on the assumption that service providers or third parties try to perform tracking or analytics with data that is actually submitted in traffic. Other data suitable for tracking or analysis purposes like IP-addresses are not considered.

Moreover, it must be mentioned that the privacy proxy only focuses on URL parameters and neglected privacy impacts by the URL path. Even though this was observed very little, breaches still can occur on the URL path. One possible solution to also increase the privacy in URL paths would be to use lists which contain path patterns that are known to leak data (similar to the lists containing known T&A hostnames). Requests whose URL paths match those patterns then could be terminated.

Another obfuscation mechanism implemented in the proxy was the blurring of location data. Apple's current iOS version offers a similar feature ("precise location", also see figure

3.1). Nevertheless, the provided method at the proxy still could improve privacy for phones running older versions of iOS. Also, the imprecise location is not applied to apps where the location permission was granted prior to iOS 14 – they will gain access to the precise coordinates [39]. Further, the underlying obfuscation mechanism used at the proxy can be modified and used for all kinds of obfuscation: returning a static set of coordinates, a completely random one or blur/sharpen the location arbitrarily.

8.2.2 Caching Approach

The caching approach was analyzed based on two sources of information: the log files provided by *squid* and generated flow data. The number of observed data points was lower than in previous analysis due to a modified data generation process.

squid was configured to cache as much as possible and ignore certain header-fields which try to control caching (see 7.1.2). However, many requests were still missed by the cache. One reason to explain this behavior, is that some types of requests are not easy to cache – especially ones with dynamic content like POST requests. The *squid* wiki lists some methods to handle such cases[29], but they are quite specific and beyond the scope of this work.

Nevertheless, *squid* manages to respond to requests which account for around half of all observed data points with cached responses. This in turn means that fewer data points are leaked to external parties. When considering the metric used in this work – the amount of leaked data points – indeed an increase of privacy could be stated. This is particularly visible on data points of category 3, as all of the requests containing geolocation data could be served by responses from the cache.

8.2.3 Observed Third Party Obfuscation

As has been shown, third party obfuscation only affects a small portion of the traffic. These findings go in line with the results found by Rao et al. which also found just little evidence of obfuscation performed by service providers or third parties [50]. Additionally, it should be noted that all observed obfuscation techniques could also be technical necessities. As there exists no insight into the inner workings of apps or service provider backends, one could only speculate about the reasons for these design decisions.

While in the investigated traffic the behavior of third party obfuscation wasn't observed in larger extent, this may change in future developments. When obfuscation in the form of encryption or custom file formats emerges, the performance of the proposed privacy proxy would suffer. In this case, further mechanisms to decrypt or parse data must be implemented.

8.3 Limitations

One limitation of the proposed approach is the reliance on the man-in-the-middle attack. For example, HTTP Public Key Pinning could be used to prevent such attacks. In the traffic generated by the observed apps, this behavior couldn't be seen. However, rejection of certificates issued by *mitmproxy* was observed in traffic unrelated to the usage of the investigated apps. Examples which failed consequently were authentication services for (mail-) accounts. Also, it was not possible to access the app store over the proxy.

Another limitation is that the proposed approach was only evaluated with six example apps of rather simple structure. When using other and more complex apps, the categorization is expected to be more time-consuming and error-prone due to the larger quantity of data

[29] see `https://wiki.squid-cache.org/ConfigExamples/DynamicContent/Coordinator`

points that must be classified according their impact on privacy. The presence of many keys would also likely increase the time required to adjust the classification in case an app loses functionality.

9 Conclusion

9.1 Conclusion

Privacy measures that are currently offered by smartphone OS providers only yield insufficient privacy protection. They mainly rely on permission management systems which have multiple shortcomings.

To overcome these flaws, different approaches to increase user's privacy exist. This work has looked into three privacy enhancing techniques (PETs). The first technique blocks data transmission to certain hosts. Predestined for such hosts are servers of advertising and tracking companies, as they provide services the user is not primary interested in or wants to avoid in the first place. A different concept is to obfuscate data. This is a very granular approach which gives extensive control over which data points to reveal and which ones to obfuscate. In this thesis, the proxy mode called *anonymize* employs obfuscation. The final PET leverages caching. When a request is answered from a cache instead from an external party, said party never sees the request which again increases the user's privacy.

All the named techniques can be deployed either at the smartphone (user side) or at a proxy (channel side). The weakness of user side approaches is the connection between the PET and the targeted apps – often a rooted/jailbroken device is required. When interacting at the channel side, the setup is easier because only a proxy has to be set up. Therefore, the proxy approach was chosen for implementation.

Since different data points have different privacy implications, a categorization was performed based on key-value pairs. In the mode *anonymize*, the proxy obfuscates the data depending on the level of impact on privacy. Fields considered as technical are not modified at all, fields considered as potentially impacting are replaced and geolocation data is reduced in accuracy.

Evaluation of the privacy proxy was performed with six apps in three modes: a baseline without any interference, blocking of known tracking and advertising hosts and obfuscation of data points in traffic. Also, the impact of a cache on privacy was investigated.

When comparing the baseline (mode *no interference*) with the blocking of known tracking and advertising hosts, a decrease of leaked data points of categories 1 (data suitable for fingerprinting or similar) and 2 (unique identifiers) was detected. When considering the assumption that less leaked data points are equivalent to an increased privacy, this goal was achieved. This shows, that already the "naive" blocking of hosts is sufficient to increase the privacy of a user.

The remaining data points which are considered under category 1 or 2 could further be obfuscated in the mode *anonymize*. Obfuscation hereby aims to replace actual values with dummy values. This further increases user's privacy by only revealing meaningless data to external parties. Accuracy of geolocation data (category 3) could also be successfully reduced.

One of the goals was to retain basic functionality of apps. The initial categorization of data points and the resulting obfuscation of data led to some broken functionality. Consequently, modifications to the categorization had to be performed in order to regain the functionality. This increases the possibility of PI leaks, but only few modifications were necessary and a potential loss of some privacy is considered justifiable to regain functionality.

The fact that functionality of apps is retained even when replacing a majority of values with dummy values, reveals the amount of data that is collected by service providers and other third parties which is not required for service offering.

Finally, a look into the possibility to introduce caching in the smartphone context is taken. Experiments showed that indeed around half of requests could be answered with cached content in subsequent runs. The effects on privacy are similar: around half of all data points considered of category 1 and 2 were caught by the cache. An exception was geolocation data, which was completely caught by the cache and therefore not leaked to external parties in the second run.

To conclude, each of the PETs enhanced the privacy of the user. The largest improvement is achieved by combining all methods: the blocking of known tracking and advertising hosts, anonymization of the remaining data points and the caching approach to reduce requests to external parties in general.

During the experiments and development of the PET, several technical issues rose. The search for a suitable caching approach was quite challenging and different architectures (and underlying software) were examined. Another obstacle was the general lack of insight. The inner working of apps and backends were considered as black boxes. While it is always hard to get insights into backends, usually it is possible to reverse engineer the locally installed apps. However, the reverse engineering process is rather complex for iOS applications due to the less open environment ([16]) and therefore this task was considered to be beyond the scope of this work.
Apps, backends, changing content provided by service providers and communication patterns of iOS were all moving parts that had influence on the reproducibility of experiments. While some disturbances could be reduced, it is impossible to eliminate all noise. However, a general trend can be seen in the provided analysis.

9.2 Future Work

Future development of the proposed privacy proxy should focus on increasing the user-friendliness of the system. While the setup of the infrastructure and the installation of the required certificates is straight-forward, the management of the privacy categorization is not.

One solution to provide end-users a convenient way to obtain categorizations would be the same as is done by the DNS sinkhole *pi-hole*. It relies on blacklists assembled by users from a community. A similar approach can be considered for providing categorizations for data points.

When assessing multiple apps from different developers, it is expected that the privacy categorizations diverge to some extent. To keep satisfying privacy elevations, the approach chosen by Ren et al. [50] could be employed: working with different privacy categorizations

dependent on the target domain. When using this approach, a more precise differentiation on the privacy impact of data points could be performed.

Also, a mechanism to whitelist known hosts which will not work under any man-in-the-middle interference (like the connection to Apple's app store) should be implemented. *mitmproxy* already provides such a feature: *ignore_hosts*. It accepts a list of regular expressions and matches those against encountered requests. When there is a match, the according request will not be intercepted and just forwarded.
As the previous proposal, these whitelists could also be managed by a community.

Additionally, the transition of the system currently running as a WiFi access point to a VPN-based system should be assessed. This would allow to use the privacy proxy also in mobile networks. From a technical point of view, nothing should hinder this shift and it was performed before [18] [48] [50].

The approach that was presented in this thesis focuses on increasing the privacy in observed HTTP traffic. It however does not try to hide network related information like IP addresses (which for example can be used for location targeting [51]) or round-trip times (which e.g. could be a part of a fingerprint). Also, the privacy threat posed by other parties like internet service providers (ISPs) – which have broad insight in one's online activities [45] – is not considered.
To additionally increase the privacy in these aspects, one could employ further PETs at the privacy proxy. An example for such a mechanism is the usage of The Onion Router (TOR). It is a network of virtual tunnels that allows to improve privacy and security on the internet [56]. This is achieved by encrypting the traffic and then sending it through multiple servers before requesting the intended content. Forwarding of encrypted traffic blocks close entities (like ISPs) to inspect the traffic. As the target server is contacted from a different node, the IP address of the TOR user also stays hidden. To enhance the privacy proxy to also cover the named aspects, it must be configured to automatically route all its traffic via the TOR network.

A Appendix

A.1 Growth of Data Leaks in Recent Years

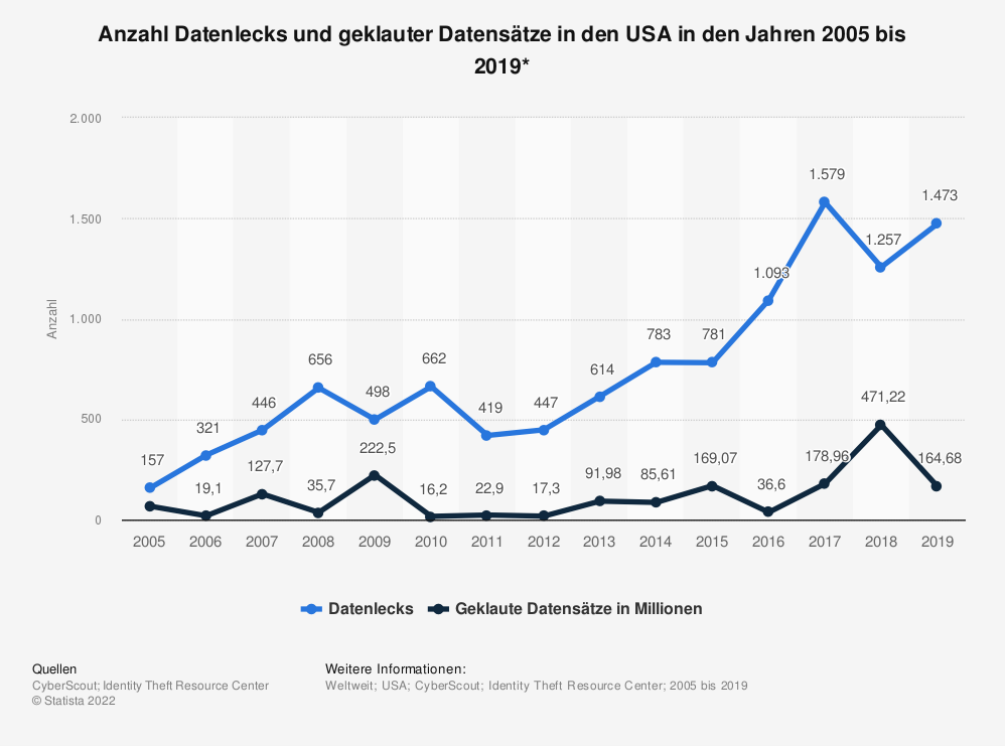


Figure A.1: Number of data leaks (blue) and number of stolen data sets (in millions, black) over the last years [28]

A.2 Defined Usage Patterns

ZDFheute	**Tagesschau**	**NTV**
Open App	Open App	Open App
Click 1st Article	Click 1st Tile	Open "Börsenticker"[30]
Scroll to Bottom	Scroll to Bottom	Click Tab "Das neueste"
Click 2nd Article	Click 2nd Tile	Click 1st Article
Scroll to Bottom	Scroll to Bottom	Scroll Until First Ad is Visible
Click "Ticker"; Don't Scroll	Click Button "News"	Click Tab "Meistgelesen"
Open Tab "Stories"; Go through 1st Story	Click on 3rd Article	Click 1st Article
Open Tab "TV"; Play Main Video for 3s	Scroll to Bottom	Scroll Until First Ad is Visible
	Click Button "TV"; Watch Livestream for 3s	Click Button "Live TV"; Watch 3s
		Click Button "Wetter"
		Scroll to Bottom

Table A.1: Defined usage patterns for each app of the news-category

wetter.com	**WetterOnline**	**Regenradar**
Open App	Open App	Open App
Popup "Hyperlokales Wetter": Select No	Search for Place: Köln	Search for Place: Memmingen
Search for place: Berlin	Click on "90 Min-Wetter"	Wait Until Map is Loaded
Scroll to Bottom	Scroll to Bottom	Scroll the Map Towards Berlin
Open Menu; Enable "automatischer Ort"	Click "Search"	OS-Tray: "Allow Access to Location Once"
OS-Tray: "Allow Access to Location Once"	OS-Tray: "Allow Access to Location Once"	Wait Until Map is Loaded
Scroll to Bottom	Scroll to Bottom	Zoom on Location Pin As Far As Possible

Table A.2: Defined usage patterns for each app of the weather category

[30] Wait until content is loaded, but at max 5 s

A.3 Caching Details

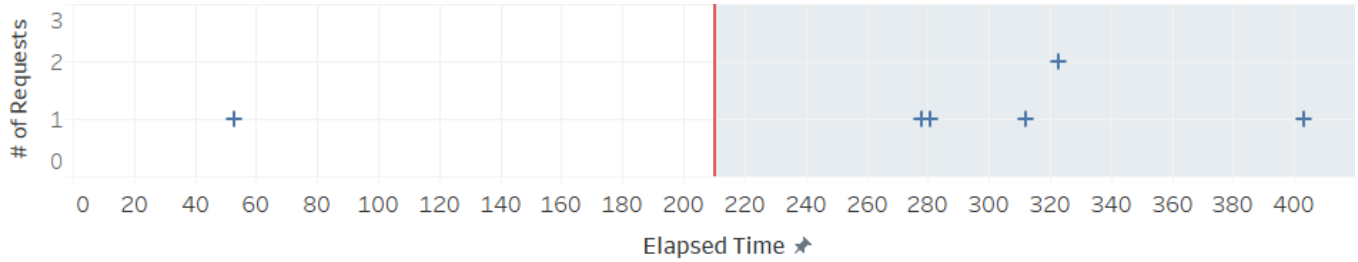


Figure A.2: Number of cache hits on objects stored on disk. It can be seen that in the second (shaded) run, more hits were observed. However, the overall number of disk hits is rather low.

List of Figures

List of Tables

Bibliography

[1] AdAway. *AdAway Wiki. HostsSyntax*. June 2013. URL: https://github.com/AdAway/AdAway/wiki/HostsSyntax. accessed on 05.04.2022.

[2] S. Amini, J. Lindqvist, J. Hong, M. Mou, R. Raheja, J. Lin, N. Sadeh, and E. Toch. "Caché: caching location-enhanced content to improve user privacy". In: *Mobile Computing and Communications Review* 14 (Dec. 2010), pp. 19–21. DOI: 10.1145/1923641.1923649.

[3] K. E. Anderson. "Ask me anything: what is Reddit?" In: *Library Hi Tech News* 32.5 (Jan. 2015), pp. 8–11. ISSN: 0741-9058. DOI: 10.1108/LHTN-03-2015-0018.

[4] Apple. *Accessing User Data and Resources*. URL: https://developer.apple.com/design/human-interface-guidelines/ios/app-architecture/accessing-user-data/. accessed on 26.01.2022.

[5] Apple. *Entitlements*. URL: https://developer.apple.com/documentation/bundleresources/entitlements. accessed on 27.03.2022.

[6] Apple. *Support pages. If an app asks to track your activity*. Apr. 2021. URL: https://support.apple.com/en-us/HT212025. accessed on 29.09.2021.

[7] Apple Inc. *Apple Support Pages. Use Low Data Mode on your iPhone and iPad*. Dec. 2020. URL: https://support.apple.com/en-us/HT210596. accessed on 19.03.2022.

[8] A. Arampatzis. *What Is the Difference between Root Certificates and Intermediate Certificates?* July 2020. URL: https://www.venafi.com/blog/what-difference-between-root-certificates-and-intermediate-certificates#:~:text=A%20root%20certificate%20is%20a,Intermediate%20Certificate.. accessed on 29.03.2022.

[9] B. Auxier, L. Rainie, M. Anderson, A. Perrin, M. Kumar, and E. Turner. *Americans and Privacy: Concerned, Confused and Feeling Lack of Control Over Their Personal Information*. Pew Research Center. Nov. 2019. URL: https://www.pewresearch.org/internet/2019/11/15/americans-and-privacy-concerned-confused-and-feeling-lack-of-control-over-their-personal-information/.

[10] S. Black. *Unified hosts file with base extensions*. Mar. 2022. URL: https://github.com/StevenBlack/hosts. accessed on 10.03.2022.

[11] Cambridge University Press. *Cambridge Dictionary*. URL: https://dictionary.cambridge.org. accessed on 27.01.2022.

[12] M. Chen, W. Li, Z. Li, S. Lu, and D. Chen. "Preserving location privacy based on distributed cache pushing". In: *2014 IEEE Wireless Communications and Networking Conference (WCNC)*. 2014, pp. 3456–3461. DOI: 10.1109/WCNC.2014.6953141.

[13] Cloudflare. *What is HTTPS?* URL: https://www.cloudflare.com/de-de/learning/ssl/what-is-https/. accessed on 20.04.2022.

[14] CookiePro. *What is a Tracking Pixel?* Sept. 2021. URL: https://www.cookiepro.com/knowledge/tracking-pixel/. accessed on 12.04.2022.

[15] M. Cunha, R. Mendes, and J. P. Vilela. "A survey of privacy-preserving mechanisms for heterogeneous data types". In: *Computer Science Review* 41 (2021), p. 100403. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2021.100403.

[16] J. Damian. *Basic iOS Mobile App Reverse Engineering*. Sept. 2021. URL: https://www.nowsecure.com/blog/2021/09/08/basics-of-reverse-engineering-ios-mobile-apps/. accessed on 18.04.2022.

[17] dpunkt. Verlag. *Squid Handbuch. Squid Result Codes*. 2003. URL: http://www.squid-handbuch.de/hb/node103_ct.html#Squid-result-codes. accessed on 17.03.2022.

[18] S. Elvery. *Data Life. Server Setup*. Oct. 2018. URL: https://github.com/abcnews/data-life/tree/master/server. accessed on 08.11.2021.

[19] S. Englehardt. *Firefox 72 blocks third-party fingerprinting resources*. Jan. 2020. URL: https://blog.mozilla.org/security/2020/01/07/firefox-72-fingerprinting/. accessed on 18.04.2022.

[20] European Commission. *What is personal data?* URL: https://ec.europa.eu/info/law/law-topic/data-protection/reform/what-personal-data_en. accessed on 20.10.2021.

[21] European Union Agency For Network and Information Security. "Privacy and data protection in mobile applications. A study on the app development ecosystem and the technical implementation of GDPR". In: (Jan. 2018).

[22] K. Fawaz and K. G. Shin. "Location Privacy Protection for Smartphone Users". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS '14. Scottsdale, Arizona, USA: Association for Computing Machinery, 2014, pp. 239–250. ISBN: 9781450329576. DOI: 10.1145/2660267.2660270.

[23] A. Fox. *Precision Matters: The Critical Importance of Decimal Places*. Sept. 2017. URL: https://blis.com/precision-matters-critical-importance-decimal-places-five-lowest-go/. accessed on 20.04.2022.

[24] J. Golbeck. "Chapter 9 - Twitter". In: *Introduction to Social Media Investigation*. Ed. by J. Golbeck. Boston: Syngress, 2015, pp. 85–100. ISBN: 978-0-12-801656-5. DOI: https://doi.org/10.1016/B978-0-12-801656-5.00009-3.

[25] Google. *HTTPS encryption on the web*. URL: https://transparencyreport.google.com/https/overview?hl=en. accessed on 22.01.2022.

[26] Google. *Permissions on Android*. Jan. 2022. URL: https://developer.android.com/guide/topics/permissions/overview. accessed on 26.01.2022.

[27] Google. *Play Console Help. Advertising ID*. URL: https://support.google.com/googleplay/android-developer/answer/6048248. accessed on 29.09.2021.

[28] Identity Theft Resource Center. *Anzahl Datenlecks und geklauter Datensätze in den USA in den Jahren 2005 bis 2019*. Statista GmbH. 2020. URL: https://de.statista.com/statistik/daten/studie/865084/umfrage/anzahl-datenlecks-und-geklauter-datensaetze-in-den-usa/. accessed on 21.01.2022.

[29] Internet Engineering Task Force (IETF). *Hypertext Transfer Protocol (HTTP/1.1). Request*. July 1999. URL: https://datatracker.ietf.org/doc/html/rfc2616#section-5. accessed on 09.04.2022.

[30] Internet Engineering Task Force (IETF). *Hypertext Transfer Protocol (HTTP/1.1): Caching. Cache-Control*. July 2014. URL: https://datatracker.ietf.org/doc/html/rfc7234#section-5.2. accessed on 08.04.2022.

[31] Internet Engineering Task Force (IETF). *Internet Content Adaptation Protocol (ICAP). RFC 3507*. Apr. 2003. URL: https://datatracker.ietf.org/doc/rfc3507/. accessed on 21.04.2022.

[32] Internet Engineering Task Force (IETF). *The Cache-Status HTTP Response Header Field*. Aug. 2021. URL: https://datatracker.ietf.org/doc/draft-ietf-httpbis-cache-header/10/. accessed on 20.04.2022.

[33] Internet Engineering Task Force (IETF). *Uniform Resource Identifier (URI): Generic Syntax*. Jan. 2005. URL: https://datatracker.ietf.org/doc/html/rfc3986#section-3.4. accessed on 20.04.2022.

[34] M. Isaac. *Uber's C.E.O. Plays With Fire. Travis Kalanick's drive to win in life has led to a pattern of risk-taking that has at times put his ride-hailing company on the brink of implosion.* Apr. 2017. URL: https://www.nytimes.com/2017/04/23/technology/travis-kalanick-pushes-uber-and-himself-to-the-precipice.html. accessed on 30.03.2022.

[35] N. Kaaniche, M. Laurent, and S. Belguith. "Privacy enhancing technologies for solving the privacy-personalization paradox: Taxonomy and survey". In: *Journal of Network and Computer Applications* 171 (Aug. 2020), p. 102807. DOI: 10.1016/j.jnca.2020.102807.

[36] B. Krishnamurthy and C. E. Wills. "Generating a Privacy Footprint on the Internet". In: *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*. IMC '06. Rio de Janeriro, Brazil: Association for Computing Machinery, 2006, pp. 65–70. ISBN: 1595935614. DOI: 10.1145/1177080.1177088.

[37] E. Laziuk. *iOS 14.5 Opt-in Rate - Daily Updates Since Launch*. Apr. 2021. URL: https://www.flurry.com/blog/ios-14-5-opt-in-rate-att-restricted-app-tracking-transparency-worldwide-us-daily-latest-update/. accessed on 13.10.2021.

[38] Linux man-pages project. *Linux Programmer's Manual. hosts*. Mar. 2021. URL: https://man7.org/linux/man-pages/man5/hosts.5.html. accessed on 05.04.2022.

[39] B. Mayo. *iOS 14 lets users grant approximate location access for apps that don't require exact GPS tracking. How to choose approximate or exact location permission?* Aug. 2020. URL: https://9to5mac.com/2020/08/12/ios-14-precise-location/. accessed on 22.03.2022.

[40] J. Mazel, R. Garnier, and K. Fukuda. "A comparison of web privacy protection techniques". In: *Computer Communications* 144 (2019), pp. 162–174. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2019.04.005.

[41] mitmproxy. *How mitmproxy works*. URL: https://docs.mitmproxy.org/stable/concepts-howmitmproxyworks/. accessed on 26.01.2022.

[42] mitmproxy. *Introduction*. URL: https://docs.mitmproxy.org/stable/. accessed on 02.02.2022.

[43] J. Mylopoulos, L. Chung, and E. Yu. "From Object-Oriented to Goal-Oriented Requirements Analysis". In: *Commun. ACM* 42.1 (Jan. 1999), pp. 31–37. ISSN: 0001-0782. DOI: 10.1145/291469.293165.

[44] I. Nai Fovino, R. Neisse, D. Geneiatakis, and I. Kounelis. "Mobile Applications Privacy. Towards a methodology to identify over-privileged applications". In: (2014).

[45] A. O'Driscoll. *Your ISP can see your browsing history; here's how to stop it*. Jan. 2022. URL: https://www.comparitech.com/blog/vpn-privacy/stop-isp-tracking-browsing-history/%5C#:~:text=Internet%20Service%20Providers%20(ISPs)%20can,using,%20and%20your%20geographic%20location.. accessed on 14.04.2022.

[46] V. Patel. *GetSocial.im Blog. Device Fingerprinting for Mobile Attribution*. Jan. 2021. URL: https://blog.getsocial.im/device-fingerprinting-for-mobile-attribution/. accessed on 25.03.2022.

[47] J. Pennekamp, M. Henze, and K. Wehrle. "A survey on the evolution of privacy enforcement on smartphones and the road ahead". In: *Pervasive and Mobile Computing* 42 (Sept. 2017). DOI: 10.1016/j.pmcj.2017.09.005.

[48] A. Rao, A. M. Kakhki, A. Razaghpanah, A. Tang, S. Wang, J. Sherry, P. Gill, A. Krishnamurthy, A. Legout, A. Mislove, and D. Choffnes. "Using the Middle to Meddle with Mobile". In: 2013.

[49] A. Rao, J. Sherry, A. Legout, A. Krishnamurthy, W. Dabbous, and D. Choffnes. "Meddle: middleboxes for increased transparency and control of mobile traffic". In: Dec. 2012, pp. 65–66. DOI: 10.1145/2413247.2413286.

[50] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes. "ReCon: Revealing and Controlling PII Leaks in Mobile Network Traffic". In: *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '16. Singapore, Singapore: Association for Computing Machinery, 2016, pp. 361–374. ISBN: 9781450342698. DOI: 10.1145/2906388.2906392.

[51] A. Schiff. *Your Quick And Dirty Guide To IP Address Targeting After Apple's Declaration Of War*. June 2021. URL: https://www.adexchanger.com/privacy/your-quick-and-dirty-guide-to-ip-address-targeting-after-apples-declaration-of-war/. accessed on 14.04.2022.

[52] Similarweb Ltd. *Top-Apps-Ranking. Filters: Apple App Store, Germany, Weather, Top Free*. Nov. 2021. URL: https://www.similarweb.com/de/apps/top/apple/store-rank/de/weather/top-free/iphone/. accessed on 03.11.2021.

[53] Similarweb Ltd. *Top-Apps-Ranking. Filters: Apple App Store, Germany, News, Top Free*. Nov. 2021. URL: https://www.similarweb.com/de/apps/top/apple/store-rank/de/news/top-free/iphone/. accessed on 03.11.2021.

[54] squid. *Squid Documentation. Squid configuration directive refresh_pattern*. URL: http://www.squid-cache.org/Doc/config/refresh_pattern/. accessed on 22.03.2022.

[55] The Zeek Project. *Zeek Documentation. About Zeek*. Mar. 2022. URL: https://docs.zeek.org/en/master/about.html. accessed on 03.04.2022.

[56] Tor Project. *Tor Browser User Manual. About Tor Browser*. URL: https://tb-manual.torproject.org/about/. accessed on 23.03.2022.

[57] J. van Hoboken and R. Ó. Fathaigh. "Smartphone platforms as privacy regulators". In: *Computer Law & Security Review* 41 (2021), p. 105557. ISSN: 0267-3649. DOI: https://doi.org/10.1016/j.clsr.2021.105557.

[58] WHATWG (Apple, Google, Mozilla, Microsoft). *URL Living Standard. application/x-www-form-urlencoded*. Feb. 2022. URL: https://url.spec.whatwg.org/#application/x-www-form-urlencoded. accessed on 15.03.2022.

[59] S. Wu and J. Liu. "Overprivileged Permission Detection for Android Applications". In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. 2019, pp. 1–6. DOI: 10.1109/ICC.2019.8761572.

[60] V. Zhou. *A Simple Explanation of the Bag-of-Words Model*. Nov. 2019. URL: https://victorzhou.com/blog/bag-of-words/. accessed on 04.04.2022.