Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Exposure of Worker Activity Times Through Application Software and Its Limitation

Kumulative Dissertation

zur Erlangung des akademischen Grades

Dr. rer. nat.

an der Fakultät für Mathematik, Informatik und Naturwissenschaften der Universität Hamburg

eingereicht beim Fachbereich Informatik von

Christian Burkert
geboren in Crailsheim

Oktober 2022

Gutachter:
Prof. Dr. Hannes Federrath
Prof. Dr. Dominik Herrmann

Tag der Disputation:
16. Februar 2023

Danksagung

Zunächst geht der Dank an meinen Doktorvater Hannes Federrath, der zur rechten Zeit am rechten Ort war und mir die wunderbare Gelegenheit gegeben hat, dort zu forschen, wo meine Überzeugung war. Danke für dieses große Vertrauen!

Ich danke Max Blochberger für seine große Geduld und Neugier beim gemeinsamen Schärfen meiner unfertigen Gedanken. Ich danke Erik Sy für seine einmalige Art, mich und meine Ideen herauszufordern. Für die vielen Teerunden und höchst inspirierenden Ablenkungen danke ich vor allem Matthias Marx, Monina Schwarz, Johanna Ansohn McDougall und Pascal Wichmann.

Für die ungemein spannende und vielfältige gemeinsame Publikation von wissenschaftlichen Beiträgen möchte ich mich bedanken bei Johanna Ansohn McDougall, Max Blochberger, Daniel Demmler, Mathias Fischer, Dominik Herrmann, Matthias Marx, Tobias Müller, Tom Petersen, Monina Schwarz, Erik Sy und Ephraim Zimmer.

Auch möchte ich mich bei allen weiteren Kolleg:innen bedanken, mit denen ich meine Zeit im 5. Stock des Gebäude F teilen durfte: Britta Böhm, Doganalp Ergenç, Steffen Haas, Malte Hamann, David Jost, Anne Kunstmann, Kevin Köster, Jens Lindemann, Sadaf Momeni, Henning Pridöhl, Eugen Ruppert, August See, Nurefşan Sertbaş Bülbül, Bahareh Shojaie, Joshua Stock, Marius Stübs, Jens Wettlaufer, Pascal Wichmann, Florian Wilkens sowie Tatjana Wingarz. Die Gespräche im Flur und später in unserer virtuellen Teeküche waren mir immer eine große Hilfe und Bereicherung.

Mein besonderer Dank geht an meine Familie und Freunde, die immer Verständnis dafür hatten, dass mein Kopf so häufig wo anders war als mein Körper und die nicht aufgegeben haben, mich immer wieder in ihre Welt zu holen. Meinen Eltern danke ich dafür, mich meinen eigenen Weg haben gehen zu lassen, aber stets in der Nähe geblieben zu sein.

Abschließend danke ich von Herzen meiner Freundin für ihren liebevollen Beistand, ihre Rücksichtnahme und ihren Ansporn während der Schlussphase der Promotion.

Zusammenfassung

Die Digitalisierung der Arbeitswelt hat zur Folge, dass ein wesentlicher Teil der Arbeitsschritte eines Beschäftigten die Interaktion mit Software (Apps) beinhaltet. Diese Interaktionen hinterlassen digitale Spuren der Beschäftigten. Sie umfassen häufig sekundengenaue Angaben über den Zeitpunkt der Interaktion (Zeitstempel). Metadaten über digitale Arbeitsschritte von Beschäftigten werden bereits zur Unternehmensoptimierung ausgewertet. Aber auch eine automatisierte Leistungskontrolle wäre technisch dadurch möglich. Aus rechtlicher Sicht stehen dem legitimen Interesse eines Arbeitgebers an der Optimierung seiner Geschäftsabläufe schutzwürdige Interessen der Beschäftigten entgegen. Die Verarbeitung von Daten über Beschäftigte unterliegt, wie personenbezogenen Daten generell, dem Datenschutzrecht. Demzufolge müssen Arbeitgeber Grundsätze wie Zweckbindung und Datenminimierung beachten, deren Einhaltung nachweisen und durch geeignete technische Maßnahmen sicherstellen (Privacy-by-Design).

Die Gefahr, die durch die Preisgabe von Aktivitätszeiten durch Software-Prozesse für die informationelle Selbstbestimmung ausgeht, wurde technisch bisher wenig untersucht. Auch gab es bisher keine Ansätze, wie die Datenschutzkonformität von Apps im Bezug auf Aktivitätszeiten untersucht, verbessert und gewahrt werden kann. Die vorliegende Dissertation umfasst Arbeiten, die zur Beantwortung dieser Fragen Beiträge liefern, um zu verstehen, welche Aufgaben Zeitstempel sowohl programmatisch als auch informativ übernehmen, welcher Bedarf seitens der App-User besteht und wie die Preisgabe reduziert werden kann.

Mittels Programmanalysen im Rahmen von App-Fallstudien kann diese Arbeit erstmals systematisch ein Potential für die Datenminimierung bei Zeitstempeln herleiten und in diesem Zusammenhang das Phänomen des übermäßigen Zeitstempelns beschreiben. Wie dieses Übermaß vermieden werden kann, beschreibt die Arbeit nicht nur konzeptionell, sondern implementiert auch ein quelloffenes Framework mit datensparsameren Zeitstempel-Alternativen. Zudem liefert diese Arbeit erstmals empirische Hinweise zum Präzisionsbedarf bei informativen Zeitstempeln und Maßzahlen für den Datenschutz-Effekt von Präzisionsreduktionen. In experimentellen Sicherheitsanalysen konnte die Arbeit erstmals den großen Umfang einer Schutzlücke bei der Verwendung von VPNs in WLAN-Hotspots nachweisen, die zur unerwünschten Preisgabe von Aktivitätsmetadaten führen kann. Abschließend stellt diese Arbeit ein neues Verfahren vor, mit dessen Hilfe Angaben für die Datenschutz-Compliance aus App-Datenmodellen generiert werden können. Dabei wird erstmals die Problematik der ubiquitären Identifizierbarkeit systematisch beschrieben und Ansätze zu dessen Abmilderung mittels neuer graphbasierter Interpretationen des Datenmodells entwickelt.

Diese Dissertation erschließt die Problematik der Preisgabe von Aktivitätszeiten in Apps. Sie zeigt auf, wie mangelnde Datenminimierung in Apps untersucht und ausgeräumt werden kann und somit die informationelle Selbstbestimmung von App-Usern gestärkt wird.

Abstract

The digitalization of the workplace means that a significant proportion of an employee's work involves interaction with software (apps). These interactions leave digital traces of the employee. They often include second-by-second information about the time of the interaction (timestamps). Metadata about employees' digital work steps is already being evaluated for business optimization. But automated performance monitoring would also be technically possible. From a legal point of view, the legitimate interest of an employer in optimizing its business processes is opposed by the interests of employees that are worthy of protection. The processing of employee data, like personal data in general, is subject to data protection laws, Accordingly, employers must observe principles such as purpose limitation and data minimization, demonstrate compliance and ensure it through appropriate technical measures (privacy by design).

The danger posed to informational self-determination by the exposure of activity times by software processes, has been little investigated technically. Nor have there been approaches to investigate, improve, and preserve the privacy compliance of apps with respect to activity times. This dissertation includes work that contributes to answering these questions, to understand what tasks timestamps perform both programmatically and informatively, what needs exist on the part of app users and how the exposure can be reduced.

Using program analyses in the context of app case studies, this thesis can for the first time systematically deduce a potential for data minimization in timestamps, and in this context describe the phenomenon of excessive timestamping. How this excess can be avoided is not only conceptually described in the thesis, but also implemented as an open-source framework with more data-minimal timestamp alternatives. In addition, this work provides the first empirical evidence on the precision requirements of informative timestamps and measures of the privacy effect of precision reductions. In experimental security analyses, this work was able to demonstrate for the first time the large scale of a protection gap when using VPNs in Wi-Fi hotspots, which can lead to the unwanted exposure of activity metadata. Finally, this work presents a new method for generating privacy compliance information from app data models. In doing so, the problem of ubiquitous identifiability is systematically described for the first time, and approaches to mitigate it using new graph-based interpretations of the data model are developed.

This dissertation opens up the problem of activity time exposure in apps. It shows how lack of data minimization in apps can be investigated and eradicated, thus strengthening the informational self-determination of app users.

Contents

I

Introduction & Background

1 Introduction

The digitalisation of our workplaces is a story of centralisation of data, of faster data integration, and of more recording of human interaction with software. An early promise of the digitalisation of work was the promise of a paperless office [SH03]. As a result, workers were taught to no longer print out a memo, but send it via email. Taught to no longer print out a colleague's draft for editing, but to write their corrections and suggestions in the digital document. That way, data that once was ephemeral and beyond the scope of digital reach entered the digital record: the recipients of the memo and the time it was sent were recorded in the email system, and the content and time of colleagues' feedback were recorded in the exchanged text documents.

A more recent promise was the promise of centralised services [DM12]: instead of interpreting the memo at the water cooler in small groups, colleagues were provided business chat software like Slack, Microsoft Teams or Mattermost, where the entire team can discuss with a lower communication delay. And instead of exchanging text documents for review, colleagues are provided with collaborative text editing software like Google Docs or Microsoft Office 365 where they can give feedback and continue writing at the same time.

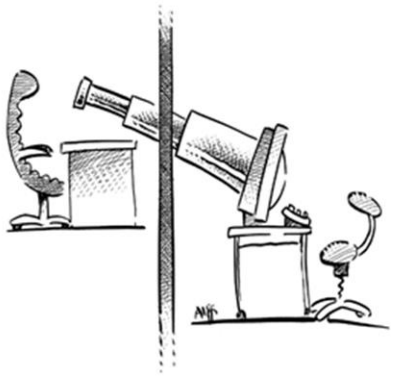
As a result of centralisation, further data that once existed only on selected workers' systems – like the feedback to the document draft – now resides on centralised infrastructure. Moreover, the granularity of observable transactions increased as a result of the earlier integration of partial work steps. Where once only the exchange of the finished feedback was observable, a collaborative text editor captures and transmits individual changes to ensure a consistent and responsive collaboration.

These changes significantly impacted the observability of work. Before digitalisation and centralisation observing work once needed an effort to capture ephemeral information. In 1920s' Germany, these efforts culminated in the standardisation of methods to describe and measure work processes and the allocation of work time to subprocesses. The 'Reichsausschuß für Arbeitszeitermittlung' (English: 'Reich Committee for Working Time Determination') was established to bring scientific methods to work time rationalisation and promote best practices [Böh67]. As part of the REFA method, manual time measurements of work steps were and maybe still are performed by rationalisers using stopwatches. Figure 1.1a depicts such a stopwatch in front of a standardised documentation sheet. Manual time measurements by rationalisers occur in plain sight of the workers and usually came announced and in consultation with worker representatives.

Through digitalisation and centralisation, work monitoring is now more inconspicuous. An unknown member of the office of the State Data Protection Commissioner of Schleswig-Holstein expressed this strikingly through a caricature shown in Fig. 1.1b: The tool for observation and measurement has become the very tool that workers perform or document their

tasks on. As a result, the observation can occur unnoticed and with little to no effort, thereby causing or increasing a power and information imbalance between employer and employee.

(a) Prior to digitalisation, rationalisation of work time involved manual measurements taken in plain sight and with the knowledge of workers. (Image: Wikipedia)



(b) With digitalisation, work steps are becoming observable through software, remotely and imperceptible. (Image: ULD Schleswig-Holstein)

Figure 1.1. Illustrations of the tooling for observing and measuring work time prior to and with digitalisation.

But as alluded to already, digitization and centralisation not only made observing work steps much more efficient and imperceptible, but also expanded the observable information significantly. Integrating partial work earlier into centralised software systems increases the observable interactions necessary to perform a task. Collaborative editors like Google Docs and Office 365 transmit individual edit commands by active users to integrate them into the shared document state. The resulting log of users' edits is recorded in such a high resolution that McCulley and Roussev [MR18] could utilise the timing information to reconstruct biometric typing profiles. Given that, it is not surprising that such recorded activity metadata is also usable to analyse workers temporal habits [ALS15; Cla+18; ETL11; TD18; tThi+14] and to estimate their performance [Wol21; WZ19].

A high degree of interaction with centralised software systems also exposes workers more to network-based observers that otherwise would have no knowledge of the activity. Going back to the collaborative editor example, each edit command has to be submitted over the network to be integrated into the central document. This client-server communication can – to a certain degree – be observed by any passive network observer and used to deduce behavioural user information [Pat+20]. Hence, even if employers are well-meaning and law abiding, there are external risks caused by using centralised and especially real-time collaborative software.

The intra-organisational risks from centrally aggregated metadata about worker activities are of course much more tangible. Employers and supervisors could decide to use the 'already available' metadata for secondary purposes like the aforementioned process rationalisation or even employee performance analysis. Microsoft offers with Workplace Analytics a tool that uses seemingly necessary metadata collected by their other services like Outlook or Teams to

provide suggestive metrics about the quality of work in an organisation, e. g., the attentiveness of meeting participants [Mic]. A study by Wolfie Christl [Wol21] comes to the conclusion that employees are increasingly subjected to sanctioning, performance analysis and automatic decisions on the basis of workplace data. That methods of data analysis lack transparency and methodological soundness. Designing software in such way that it collects data about user activities contributes to this problem and further exposes workers to such methods.

The conditions under which personal data can be collected by software in the first place and later used for secondary purposes are however strictly regulated under data protection laws like the EU General Data Protection Regulation (GDPR) or its preceding directive. Organisations are not at liberty to arbitrarily process data about their workers and have to ensure that workplace software is compliant. But whether workplace software is compliant is a complex question. In my dissertation project, I aimed to better the understanding how exposure of worker activity in software influences compliance and how compliance can be improved by technical means.

The remainder of this introductions lays out the problems related with activity time exposure and summarises my dissertation project with respect to research questions, methodology, and contribution.

1.1 Problem Description

The behaviour of much of modern collaborative software to effectively generate a centralised record of workers' activities is arguably a byproduct of contemporary expectations of convenience and agility in software-based work processes. But as this accumulation of data is at least partially related to identifiable natural persons (data subjects), data protection regulations like GDPR have to be obeyed to guarantee workers' right to privacy. Looking at GDPR and similar regulations, this includes fundamental obligations of *informing* the data subjects (here the workers) about the 'categories of personal data' and the purposes that they are used for (Article 14 GDPR), as well as the obligation of *data minimisation*, by which the processing of personal data shall be limited to what is necessary in relation to the stated purpose (Article 5(1)(c) GDPR).[1] While this might sound straight forward and easy at first, complying to these obligations when operating collaborative software raises several problems, not only for the operator (controller) but also for the software manufacturer:

1. Controllers have very **limited influence on the behaviour of standard software** and thus rely on the software manufacturer to provide a product that can be operated in a compliant manner.
2. Controllers have **limited insight into to inner workings** of standard software and thus rely on the software manufacturer to provide truthful and comprehensive descriptions of their software's behaviour. This of course also limits a controller's potential to make

[1] Throughout this dissertation, I will use the terms *data subject*. *controller*, *purpose*, etc. in the sense in which they are defined by Article 4 GDPR and common in data protection research.

informed choices among software competitors. It also limits a controller's ability to fulfil compliance obligations like informing about the processed personal data.

3. Software manufacturers might themselves **lack sufficient overview** about the processing of personal data in complex software products or are unable or unwilling to provide the resources for a privacy-specific quality assurance.
4. Software manufacturers have to **anticipate the application scenarios** of their customers in order to (at least) provide a suitable configuration option. This is particularly challenging with respect to the data minimisation requirement which generally prohibits the processing of personal data that is not necessary for the concrete purposes. Consequently, if a controller does not require a particular software feature which processes personal data but cannot be disabled, operating this software anyways would constitute a compliance risk[2].
5. There are **no well-established best practices** for in-depth data minimisation and software manufacturers might shy away from extra efforts that go beyond common industry practices.

Beyond the consideration of individual software, a systematic view on activity time exposure must also include the work environments and threats that originate from the underlying infrastructure. As collaborative software typically follows a client-server communication model, it also exposes the workers to observers on the network.

Tackling the exposure of activity times in apps thus requires efforts in all mentioned problem areas. To structure my research, I divided the problems into three main pillars following the phases that software or privacy engineers have to perform to reduce activity time exposure: (i) understand the sources of activity timestamping and the potential for minimisation, (ii) improve privacy by minimising and mitigating expose, and (iii) maintain compliance and transparency by monitoring app changes. The pillars are illustrated in Fig. 1.2.

1.1.1 Activity Timestamping and Minimisation Potential

The first step towards tackling the problem of activity time exposure is to understand the source of activity timestamping. This pillar includes my research on investigating the status quo of timestamp usage in application design (Chapter 3) as well as on investigating user demand for timestamp precision (Chapter 4). Understanding both is necessary to determine minimisation potentials. I use the term *potential* in the sense that a lack of apparent user demand or technical necessity removes the overall necessity for particular personal data, which as a result means that data processing can and should be further minimised.[3] An unrealized data minimisation potential should be considered a data protection compliance risk, as I will later discuss in Chapter 2.

[2] See Sect. 2.3 on the limits of data minimisation obligations.

[3] The word potential is not to be understood in an optional or discretionary sense. In fact, the minimisation is mandatory if the necessity is no longer given.

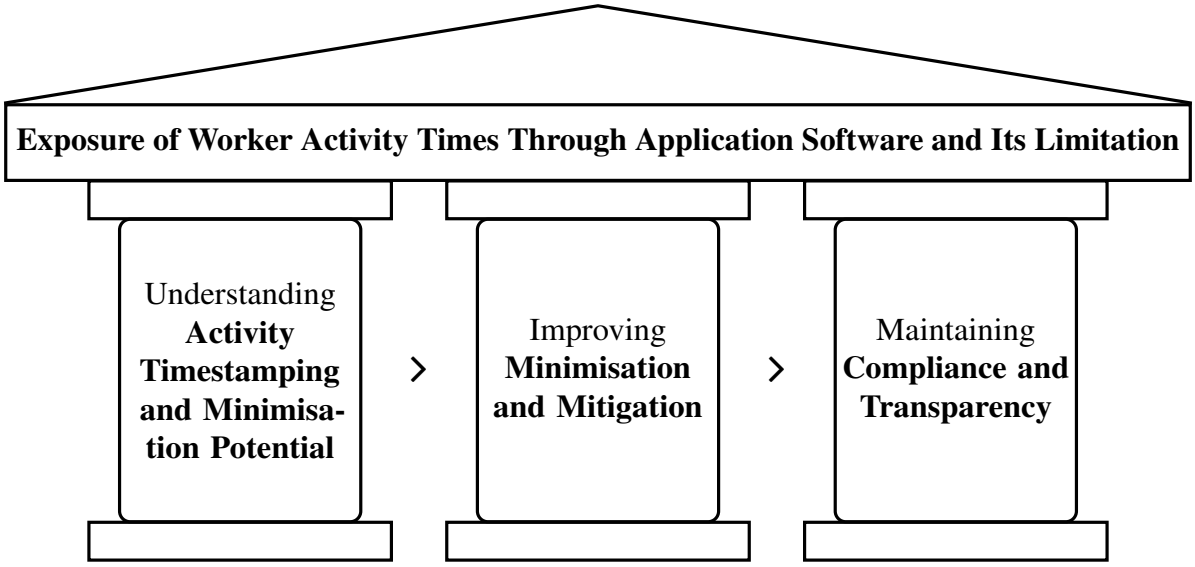


Figure 1.2. The thesis' three main pillars of researching the exposure of worker activity times.

Understanding Timestamp Usage by Apps

The fact that apps collect and store activity timestamps becomes apparent in most apps by skimming through the user interface. Their activity timestamps are commonly shown to users to provide temporal context to the surrounding information. I will focus on these timestamps in the next section. But these user-visible timestamps are generally just a subset, the literal surface of activity timestamping. Other activity timestamps are collected and not displayed but processed as input to some application logic, for instance to decide, when a session has reached its maximum duration and should be terminated. I call these *programmatic timestamps*. Sadly, there is also a third kind of activity timestamp, namely those that are collected but appear to be not used at all.[4]

Understanding the activity timestamping in apps consequently requires an analysis of general timestamp usage in apps and of the purposes they are meant to fulfil for the app. Obviously, for any timestamping to occur in an app, it had to be implemented at some point during software development. Hence, analysing timestamp usage is also a matter of understanding developers' programming habits and patterns, which can inform the design of alternative solutions to facilitate more data-minimal apps.

Method of Auditing Analysing timestamp usage has aspects of program analysis [NNH04] in the sense that once relevant timestamps have been found, automatic techniques of static program analysis can be applied to locate all references to that timestamp in the source code.

[4] I only consider timestamps as used if they are either used by logic or appear in the front end. Timestamps whose only 'use' is that they are served via an API for unspecified purposes are considered as unused.

Finding the relevant timestamps and later interpreting the located references semantically are however tasks that go beyond typical problems of program analysis and which are specific to data minimisation research. To the best of my knowledge, there existed no prior method to analyse timestamp usage in apps.

Identifying Timestamp Attributes Regarding the determination of relevant timestamps, my research generally focuses on timestamps that are recorded in app data models and thus are persisted for later processing. Finding relevant timestamps in data models raises the question of determining timestamp attributes from other attributes, which might use the same type. Depending on the programming language and frameworks data model timestamps might have a characteristic type or be of a generic, e. g., numeric type like an unsigned integer. In case of the latter, timestamp attributes have to be distinguished first by considering other characteristic features like attribute naming.

Finding the relevant timestamps further requires to determine which timestamps can be considered personal data. Narrowing down the scope to personally identifiable timestamps reduces the workload during later purpose evaluation, where analysing non-personally identifiable timestamps does not contribute to the understanding of data protection risks. Personally identifiable timestamps might be determined using the semantic context provided by data model relations, in the sense that they establish which entities are relatable to user-specific entities.

Finding Purposes Finding purposes, i. e., the aims, for which timestamps are processed implies several challenges: First of all, it seems impossible to specify and limit a-priori which type of programmatic timestamp use is significant for determining purposes. Other than in application scenarios where the possible uses are known beforehand and correspond to specific sinks in the source code, e. g., a call to a network library, the programmatic uses of timestamps seem not predictable or limitable to certain sinks. As a consequence, finding purposes needs to regard all references to the identified timestamp attributes. Due to the lack of a comprehensive prior knowledge about timestamp purposes, the purpose determination and classification has to be explorative and bottom-up.

Separating Purposes Because of the versatility of conventional timestamps it is not uncommon that a given timestamp has multiple heterogeneous uses. For instance, might the same timestamp be used to form a chronological order of items and to detect which items lie within a user-defined temporal range, e. g., a search filter. At first, these might appear to be the same purpose but considered separately, both uses might have different implications and potentials for data minimisation. In the given instance, the chronological order requires only internal ordering to make instances temporally comparable among each other, while the search filter requires comparability with external references like a human's conception of time. For that reason, a single data model timestamp might fulfil multiple programmatic uses which need to be regarded separately in the analysis.

Usage Archetypes and Alternatives Despite the versatility of conventional timestamps, it seems reasonable to expect a smaller subset of frequent programmatic uses of timestamps that commonly occur in apps. The aforementioned chronological ordering and date filtering could be examples of such common programmatic uses, which can be referred to as *usage archetypes*. These archetypes might be highly application or technology specific and can of course not be generalised or comprehensively determined by looking at few individual apps. Nonetheless, it can be expected that explorative case studies reveal fundamental archetypes usable for proposing further data minimisation measures (cf. Sect. 3.3.5).

Understanding User Demand for Timestamps and Timestamp Precision

One reason why activity timestamps are collected is certainly user information. But as a purpose in the data protection sense, 'user information' is very likely not specific enough to be compliant nor is it specific enough to function as a guideline for implementing protective measures. Removing this vagueness requires first to determine more specific purposes and second to determine what data is necessary to fulfil each purpose. Why this is challenging is explained in this section.

Vague and Unknown Usage A well-known example for timestamps with user information purpose (in the following *informative timestamps*) are e-mail timestamps. In that example, the temporal information is not used by the email app, i. e., user agent, itself to fulfil a functionality, nor by the server-side agents. The e-mail app rather collects the information to relay it to the user to contextualise a presented activity item, here, the e-mail. The interpretation of this timestamp is in the user's discretion. For e-mail, the Internet Message Format specified in RFC 5322 [Res08] does not say anything about the intended use for the *Date* field. Users could use the temporal information arbitrarily for the scheduling of their own tasks or to spy on their colleague's activity patterns. In that sense, user information is a *soft purpose* that is not as clearly as algorithmically determined purposes. From an outcome perspective, relaying data to a user resembles a raw data export without purpose specification. This power of discretion is however in contradiction to the data protection principle of *purpose limitation* (Article 5(1)(b) GDPR), which demands that personal data shall be 'collected for specified, explicit and legitimate purposes'.

The purposes for which users 'process' activity timestamps have however not been the focus of public research yet. As part of a research project[5], we asked developers of a project partner in unrepresentative, qualitative interviews, about the purposes for which they use displayed activity times. The developers, although working at the same company under presumably similar circumstances, provided a large variety of answers: Some, mentioned that they use activity timestamps to determine novelty and thus relevance of others' activities. A lead developer responded that they needed the information to effectively monitor their colleagues'

[5]EMPRI-DEVOPS https://empri-devops.de

progress. Another developer said that they did not use the information at all. This limited evidence already suggests that the purposes can be very subjective and possibly excessive.

Usefulness of Specificity Putting effort into identifying more specific purposes for informative activity timestamps might be seen as futile, given that a specified purpose is practically impossible to control once a user received the information. As there is a media breach between the app's user interface and the user, and software can neither control nor monitor what users do with the information once they learnt it, continuing to enforce a usage control policy on the user side is – at least today – science fiction.[6] But even though it is not enforceable by technical means, a controller can and arguably should implement organisational means by which the usage of activity times for other than the intended purposes, e. g., spying on colleagues, is forbidden and sanctioned. In terms of compliance, legal GDPR commentaries do not include the de facto limitation of technical enforceability as a factor to consider when determining the necessary specificity of purposes [Art13; SHS19].

Regardless of technical enforceability, a declared purpose has to be sufficiently specific to fulfil its function of providing transparency to data subjects and auditors [SHS19, Art. 5 Rn. 76ff] (cf. Sect. 2.2). I would argue that especially for informative timestamps, it is important to identify purposes that are specific enough to reason about data minimisation potentials in depth. The lack of containment at the user interface makes it impossible to assess or mitigate the impact of excessively collected personal data. In contrast, for non-informative timestamps that are processed only programmatically, a breach of purpose limitation can be determined by program analysis. In other words, for programmatically processed data, the de facto purposes are eventually committed in written code, while soft purposes have no auditable implementation. Purpose limitation and data minimisation cannot technically prevent misuse of informative timestamps, but they can limit the exposure of activity timestamps by collecting fewer and less precise timestamps.

Obtain Understanding Identifying sufficiently specific sub-purposes requires a deep understanding of the informative timestamps' functions within the work (normative approach) or of the workers' usage of the timestamps (descriptive approach). Following the normative approach of specification could make use of existing process documentations or guidelines that codify decisions that involve informative activity timestamps, e. g., an instruction to preface customer feedback with an apology if it did not come within three days. This approach relies on a comprehensive and up-to-date description of work processes which is presumably not widespread. Following the descriptive approach requires evidence of usage that could be gathered from interviews with workers or by observing their work. This approach might fall short due to an insufficient self-awareness of workers or because of the difficulty of actually observing the cognitive processing of the informative timestamp, which makes it dependant on unreliable self-reporting. Instead, indirect sources of evidence could be used, where available,

[6] The TV series 'Severance' depicts a technology that can prevent workers from accessing their work memories outside of work.

to deduce usage from observable factors. For instance, the usage of the e-mail date could be better understood by looking at workers' e-mail client preferences, specifically whether or not they have the date column visible or how they have it formatted. Assuming that half of the workers in a team chose to not have the date column visible, then this would be an indicator that the date is not necessary for the purpose of their work.

Implement Understanding Based on guidelines, interviews or indirect evidence, the controller can deduce where informative timestamps are unnecessary or what appears to be the lowest sufficient precision. But in terms of implementing this understanding in software, controllers are limited by what configuration options the software manufacturer provides to customize the processing of personal data. If the software manufacturer did not anticipate and serve the demand for customisability, controllers are left with either doing nothing and tolerating the compliance risk or carrying out costly modifications on their own, if even possible. For software manufacturers to even be able to anticipate demands, research is needed to understand human interaction with informative activity timestamps in general. Large scale analysis of documented user demand in particular apps could help to surface commonalities and inform software design decisions.

1.1.2 Minimisation and Mitigation

The second pillar is concerned with measures to improve the status quo of activity time exposure. On the one hand, I discuss technical measures to minimise timestamps in the app design by providing more data-minimal alternatives. On the other hand, I describe where further mitigation is necessary to reduce the remaining observability by network-based observers.

Minimising Timestamp Use and Precision

Our case studies of app source code have revealed that timestamps are excessively used in data models and that there is real potential for timestamp minimisation. We have seen that some timestamps are without any programmatic use and can be removed without replacement. Others are used but do not have to be timestamps in order to perform their purpose. And even for those timestamps that are needed as such, data minimisation is possible and arguably obligated. As mentioned before, applying data minimisation to timestamps should not be seen as a binary option of whether to include the timestamp or not. Instead, the question of necessity should be raised for different levels of timestamp precision. This is of course dependent on the purposes for which the timestamps are used, as we discussed earlier. Hence it eventually is up to the software developer to select an appropriate minimisation.

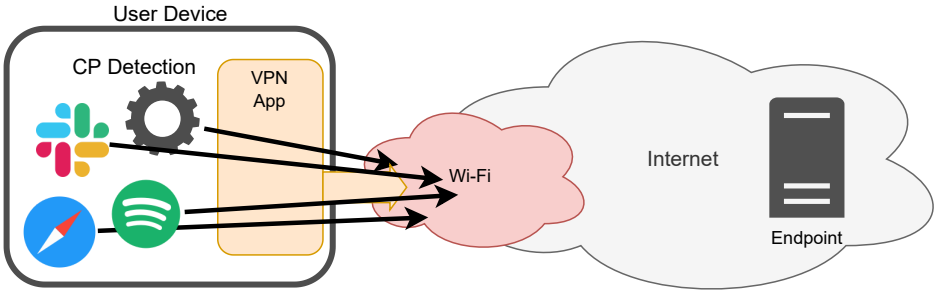
Facilitate Developers Facilitating software developers with implementing appropriate timestamp minimisation is arguably critical for the practical adoption. Conventional timestamps are ubiquitous and easy to use. They are supported by data base management systems and application frameworks with rich tool support. If developers are expected to follow data minimisation, the more privacy-preserving approaches should also come in ready-to-use frameworks.

Alternatives for Archetypes As part of our Mattermost case study (Chapter 3), we classified the timestamp purposes and suggested purpose-specific alternatives. A framework with timestamp alternatives should comprise replacements for the most common timestamp archetypes. With Mattermost, we saw the potential to use sequence and revision counters for ordering purposes and timestamps with reduced precision where external comparability is necessary. Other literature also suggests timestamps with a gradually progressing precision reduction [ZBY06]. Timestamp alternatives with reduced precision should facilitate developers with making conscious decisions about the necessary level of timestamp precision. Other than in the case of conventional timestamps, no default precision should allow developers to circumvent case-by-case decisions. If developers come to the conclusion that a 15 min precision is necessary for a given purpose, this should be explicit and deliberate during coding.

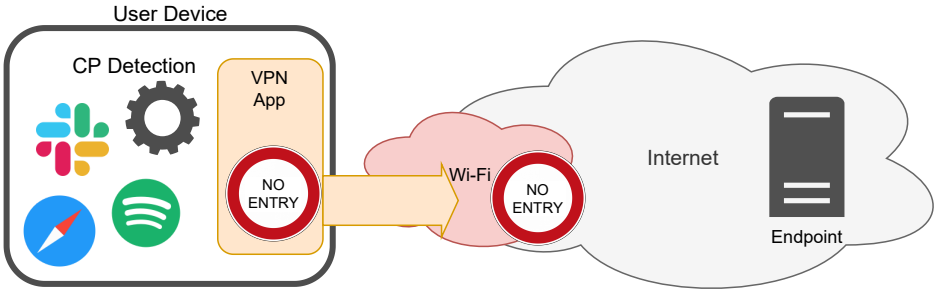
Maintain Efficiency Due to the separation of purpose, it is possible that multiple timestamp alternatives are necessary where previously a single conventional timestamp was sufficient to fulfil all purposes. For instance, the creation date of a posted message in a chat app previously might have fulfilled the function to order all posts chronologically and to enable date searches like 'since 1pm'. As described earlier, these two purposes are best minimised separately, as the first only requires ordering and no absolute frame of reference, while the second requires the opposite. The replacement in that instance could be a sequence number and a generalised date, respectively. Such separation of purpose might be possible on many occasions which would incur higher costs for the replacements compared to the original timestamp. Designs and implementations of alternatives should therefore aim to reduce such cost increases. Eventually, the resulting costs and practicality should be demonstrated for timestamp alternatives.

Observability on the Network

During the use of collaborative software, users' interactions with the software might trigger network communication with the centralised software component. For instance, if a user switches to a different channel in a business chat app, the app queries over the network for new posts. Or, if a user writes in a collaborative editor, the app continuously submits the new edits to the server. As long as the worker and the central component reside in the same trusted network, this is less of a concern. In a trusted network, users can rely on appropriate organisational and technical measures to prevent network sniffing by unauthorised persons, and sniffing by authorised persons does not increase the attack surface, as they typically have access to the communication endpoint as well.



(a) Leakage Problem: Traffic from other apps and processes can leak into the untrusted access network if a VPN app does not suppress this during its tunnel establishment.



(b) Deadlock Problem: An over-eager suppression of traffic by the VPN app can hinder the reachability of a captive portal thus causing a deadlock between the captive portal remediation and the VPN establishment.

Figure 1.3. When establishing VPN tunnels in untrusted access networks, VPN apps have to prevent leaks and allow the interaction with captive portals without deadlocks.

But as soon as the worker is mobile and communication runs over the untrusted Internet, the fine-grained communication exposes a user's activity patterns to network-based observers: Although an observer could probably not learn the contents of the communication with the central component, they can observe the time and destination, and can therefore likely infer knowledge about user activity and behaviour in the app. Moreover, Pathmaperuma et al. [Pat+20] indicate that in-app user activities could be inferred by passive eavesdroppers even from encrypted Wi-Fi traffic using machine learning classifiers.

Network-based attackers could be located at public Wi-Fis, where they, e. g., might scout potential victims for social engineering based on their network activity. For instance, the observed traffic timing could reveal the fact that the victim's employer uses GitHub and Slack based on the observed destination IP addresses and also when the victim is using them.

VPNs as Protective Measure A common protection against such attackers are Virtual Private Networks (VPNs). By using an encrypted tunnel to a VPN endpoint, network-based attackers situated in between the user and the endpoint are no longer capable to observe destinations other

than the VPN endpoint. Hence, such tunnels are a way to provide increased confidentiality in untrusted access networks like public Wi-Fis.

However, the VPN client has to establish the tunnel in a way that prevents other apps to leak traffic in the meantime. If no precautions are taken, other apps that send traffic simultaneously to the tunnel establishment could simply bypass the tunnel and be unprotected from the observer, as is illustrated in Fig. 1.3a. To avoid this, operating system and/or VPN client need to suppress other outbound traffic until and unless the outbound traffic can be routed through the established tunnel.

Interacting with Captive Networks If the access network is a public Wi-Fi network, leak prevention may be complicated by the presence of a captive portal. Captive portals are part of a mechanism to technically ensure that users of a Wi-Fi are initially directed to a website, where they are usually asked to consent to terms of service or enter credentials. Unless the user fulfils the requirement of the captive portal, they remain captive in the Wi-Fi access network and their outbound traffic is dropped.

Combining this with the expected behaviour of a VPN client, to suppress traffic until a tunnel is established, it becomes apparent, that an insufficient coordination of VPN and captive portal handling could result in deadlocks, where neither the captive portal is reachable to lift the captivity, nor the VPN endpoint is reachable to lift the blocking by the VPN client (see Fig. 1.3b). Given such a deadlock situation, a user would be forced to disable the VPN client's leak protection in order to reach the captive portal, by which they would also remove any protection against other apps leaking traffic.

Vulnerability of Mobile Workers In summary, mobile workers that use centralised collaboration tools are particularly exposed to network-based observers due to the tools' frequent network activity. VPNs employed as a mitigation need to be able to suppress other traffic prior to tunnel establishment, yet still allow requests necessary to overcome captive portals. To what extend the privacy of mobile workers is violated as a result of insufficient leakage protection of VPN apps has not been researched before.

1.1.3 Compliance and Transparency

The third and final pillar of my dissertation concerns practical challenges of monitoring and auditing data protection compliance of apps. My research here is motivated by the observation that data protection regulations like GDPR (a) put full responsibility on controllers to understand and justify the inner workings of the software they use (Article 5(2)), (b) presume comprehensive and comprehensible (for data subjects and auditors alike) documentation about the data processing (e. g., Articles 13-20, 30), and (c) fail to directly, legally oblige the one actor that has the capacity and knowledge to provide such documentation: the software manufacturer. All this makes it very challenging for controllers to fulfil their compliance

obligations for processing activities that use standard software unless the software manufacturer voluntarily supplies compliance documentation alongside their product. Without such support, understanding complex software products is beyond what is doable for most controllers. But even for the software manufacturer, keeping up-to-date compliance documentation about its own apps remains challenging, as information about processed data and its purposes must be congruent with the actual app logic throughout active development.

Avoiding the incongruency between documentation and code is perhaps one motivation why even manufacturers employ rather generic descriptions when they have to provide compliance documentation, e. g., if they inhabit the role of controller themselves as SaaS provider. For instance, GitHub confines its privacy statement regarding which usage information it collects to the following:[7]

> If you're accessing or using our Service, we may automatically collect information about how you use the Service, such as the pages you view, the referring site, your IP address and information about your device, session information, the date and time of each request, information contained in or relating to your contributions to individual repositories, and telemetry data (i.e., information about how a specific feature or service is performing) regarding your use of other features and functionality of the Service.

This statement is notably generic and could – except for the term *repository* – be describing any web application. For instance, the question 'Does GitHub retain usage information about all edits made to an issue comment including their data and time?' could not be answered with the information available in the privacy statement. At this level of abstraction, users and SaaS customers have not enough information about the concrete personal data involved to make informed decisions. Neither do auditors find enough information to check if the data processing is lawful and appropriate.

Deriving compliance documentation from the authoritative source by means of automatic routines could remove many hindrances of providing better information. The following, describes why automation can help gaining an oversight about the processed personal data and where the research problems lie.

Oversee Personal Data Processed in Apps

Getting an overview of the personal data that are processed by an app is necessary for various stakeholders in data protection:

- Software developers and quality assurance (QA) testers need to monitor the changes to ensure compliance with internal guidelines and external regulations (change monitoring).

[7] https://docs.github.com/en/site-policy/privacy-policies/github-privacy-statement (accessed 2022-08-04)

- Controllers need to continuously ensure that the software they chose is compliant and behaves in concordance with the stated processing extend and purpose. Regarding GDPR, controllers are required to document their processing activities (Article 30) including the categories of processed personal data. Moreover, they have to actively inform data subjects about the processed categories of personal data if they are not provided by the data subjects themselves (Article 14).
- Data subjects as users of an app need to be able to assess the extent of the processing to exercise their rights.
- Auditors, e. g., from data protection authorities, need to be able to examine the lawfulness of the processing based on provided documentation (Article 30).

Obtaining an Overview But where to get such an overview? The list of personal data processed by an app is not something that becomes obvious even for experienced users or administrators of the app, as they only come to know data that surfaced through the UI or other interfaces. As alluded to in the first pillar, the surfaced data might only be a subset of all collected personal data. Access to the app database and source code can provide a more complete view. A view, however, which typically extends beyond what is manageable and comprehensible by the naked eye and with manual browsing. In the apps that we analysed, data models with many dozens to hundreds of entities and hundreds to thousands of attributes were no exception. Hence, it is impractical to determine manually what in the data model is personal data.

Automatic Determination To prepare for automation, data models need to be first transformed from what is found in databases or source code into machine readable descriptions. Conventional relational database schemas contain most of the information that is needed for personal data determination, namely entity definitions in the form of tables and relation definition in the form of foreign key constraints. Together, entities and relations can be used to construct a data model graph as the basis for automatic analysis. However, some modern middleware takes over traditional database duties like relationship handling and no longer sets corresponding constraints in the database. For apps affected by such middleware, other sources of relationship information need to be researched. As a last resort, relationship information should be present in the source code, however, in less standardised form. Regardless from where the entity relationships are sourced, they should provide a structural and semantic evidence for the attribution of personal data.

Establishing Attribution For data to be considered as personal data by GDPR, essentially two conditions are required: (i) the data has to relate to a natural person, and (ii) this natural person has to be identifiable (Article 4(1)). For the purpose of app data model analysis, I only regard the first (relate) to be established explicitly through data model structure. In other words, data is relating to a natural person if the data model establishes the relationship, e. g., a chat

app's *Post* entity is personal data because it references the *User* entity as author. In contrast, if the message of a post would say 'John is sick today', this of course is also personal data (about John), but personal data that is established outside of the data model and thus not considered for the app data model analysis.

The second condition for personal data, identifiability, is usually provided by external factors beyond the scope of data models in collaborative software, e. g., the name given to a user account or other associated properties known to describe a specific natural person. As such external knowledge is very difficult to include in any automatic analysis, this work assumes identifiability of any user (or person) entity in the data model. Hence, the remaining research challenge is to establish the 'relating' condition, which we henceforth call *attribution*, a term that is less overloaded and also used by GDPR to describe this condition in the definition of pseudonymity (Article 4(5)).

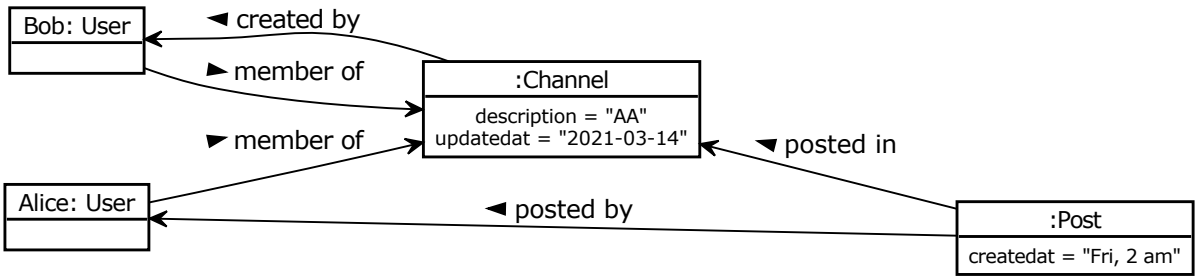


Figure 1.4. This object diagram illustrates data of a simple chat app where Alice and Bob are members of a channel that Bob created and in which Alice has written a post. Note how every attribute of a channel and post can be similarly attributed to Alice *and* Bob. This ambiguity highlights that further semantic distinction seems necessary to achieve informative personal data overviews.

Attribution Dilemma Naive approaches that determine attribution through simple directed or undirected reachability in data model graphs quickly fall short, as directed reachability ignores a significant share of relations and undirected reachability ignores semantic distinction that could help to manage what can easily turn into an overwhelming number of attributable data. To illustrate this, consider a simple example of a chat app comprising of users, channels and users' postings in channels. Figure 1.4 shows an object diagram with data instances of such an app. The sample contains the users Alice and Bob, a channel created by Bob, and a post in this channel by Alice. Approaching the question of attribution through explicit entity relations leads to the question: What is personal data about Alice and what about Bob? Putting the object diagram's content into statements, the following illustrates the ambiguity:

1. *Bob has created a channel named 'AA'.*

   Bob's creation and naming of the channel is intuitively clear as personal data about Bob and also clearly attributable to Bob through the 'created by' relation. But by way of Alice's 'member of' relation with the channel the data is also structurally attributable

to Alice. Intuitively, the creator and name of the channel provide context to Alice's membership. The data model not only contains the information that Alice is member of a channel, but also that the channel is named 'AA', the significance of which should become obvious considering that 'AA' might stand for 'Anonymous Alcoholics'.

2. *Alice is member of Bob's channel where she posted at 2 am.*

   Similarly in this statement, Alice is the person that the data intuitively relates to, but through Bob's 'created by' *and* 'member of' relation, it is also attributable to Bob. Intuitively, any channel member's behaviour might reflect on the nature of the channel and thus on its creator and all of its members. For instance, if all members tend to post after hours, then the topic of channel is likely private.

3. *The channel was last updated on March 14th.*

   Structurally and intuitively, it is unclear what and who's update this refers to. If updating is only possible for channel creators, then this additional knowledge makes it unambiguously attributable to Bob. However, this knowledge is not in the data model. Structurally, the updated time is attributable to Alice and Bob.

The examples highlight an *attribution dilemma* where strictly considering all structurally attributable data as personal data has the risk of overwhelming auditors and data subjects, while omitting attributions that seem overly broad might significantly misrepresent the data processing. Finding a good compromise between comprehensiveness and comprehensibility is a major challenge.

Interpreting Ambiguity Moreover, app data models lack the semantic context that could resolve ambiguities arising from multiple attribution paths, like the channel update time being attributable to Alice and Bob. The difference between Alice and Bob in this chat example are the roles they have. Both are members of the channel, but Bob is also the channel creator and Alice a post author. These roles, although structurally recognisable in the data model graph, are not semantically defined in the data model. Similarly, the data model lacks semantic information to resolve which role(s) a given entity attribute relates to and how (if any). For instance, what is the semantic distinction between the channel name being attributable to the channel member and the channel creator? Adding this information would involve external knowledge like a human domain expert. Requiring human input, however, limits automation. The challenge then is to design a process with minimal human involvement.

1.2 Research Questions

Having described the problem of activity timestamp exposure and its manifestation in the tree pillars of my work, this section now describes the guiding research questions of this thesis.

RQ1: How are activity timestamps used in practice by apps and for what purposes?

At the heart of this question is the exploration of a methodology or auditing process that allows to (a) identify activity timestamps in data models, and (b) determine their usage purposes. Based on that: Are there lessons to be learnt from how software developers use activity timestamps? This also involves the question if there are usage archetypes that make it possible to substitute or rather complement conventional timestamps with a practical framework of more privacy-friendly alternatives. Is there significant usage of activity timestamps for programmatic purposes that even allow for data minimisation? Are there apparent design anti-patterns that lead developers to an avoidable usage of activity timestamps?

RQ2: What is the user demand for informative activity timestamps and how can it be determined?

As a special usage type of activity timestamps, user information poses separate research questions with respect to purpose specification, purpose limitation and data minimisation, which originate from the opacity and subjectivity of human data processing: How can purposes for informative timestamps be determined more specifically? What is the user demand for timestamp precision? How can we estimate precision demand empirically? And from the other perspective: At what level would data minimisation, e. g., through timestamp precision reduction, impede the utility of informative activity timestamps? And would the acceptable levels of minimisation even be sufficient to meaningfully impact the privacy of data subjects?

RQ3: How can frameworks be designed to help developers minimise timestamp exposure?

Building on the findings about activity timestamp usage in apps and user demand for informative timestamp precision, the logical next step is to investigate technical measures that assist software developers to avoid excessive activity timestamping. The central question is: How can the lessons learnt from analysing programmatic activity timestamp usage be transformed into a framework of more privacy-preserving alternatives? What designs of alternative data types cover the identified timestamp usage archetypes? And how can the API design guide developers to a more deliberate and purpose-driven implementation? How can the designs be implemented to efficiently provide common use-case combinations and scenarios? How much effort is required to migrate existing code bases from conventional timestamps to these alternatives? What is their cost overhead, e. g., in terms of storage?

RQ4: What can mitigate the observability of mobile workers' activities by network-based attackers?

Given the typical architecture of real-time, client-server collaboration software, the observability of worker activity by the network and especially by the server is to a certain degree inevitable. The observability by the server can be mitigated by organisation measures and data minimisation. The observability by the network, however, requires additional measures. Can encrypted VPN tunnels reliably mitigate the observability? Do public Internet access points cause an additional threat of activity time exposure for mobile workers? How can traffic leakage be avoided by operating systems and VPN clients?

RQ5: How can personal data in app models be identified more automatically to monitor compliance?

A necessary prerequisite of monitoring an app's activity timestamping behaviour is to track personally identifiable timestamps in the data model. Determining, what is personal data in an app appears, at first, to be simple where the identifiability of the data subject is a direct consequence of the relationships within a data model. However, for data models of real-world complexity, this can be demonstrated to become much more complicated due to data models' high degree of cross-linking. Moreover, the differences in relation direction and multiplicity of relation paths require interpretation with regard to identifiability and particularly attribution. How can app data models be used to determine personal data in general? How do model entity relationships influence the propagation of data subject attributability? How can the complexity of large and dense data models be handled? Can data subject categories be determined based on attribution paths and how can these be found more efficiently for large graphs? And more practically, how can developers, QA testers and reviewers use this data model analysis in continuous processes and more automatically?

1.3 Research Methodologies

This section provides an overview about the research methodologies that I used in my work to find answer to the previously described research questions. Addressing the issues of activity time exposure at all pillars (understanding, improving and maintaining) requires a broad spectrum of methods, both analytical and constructive.

1.3.1 Program Analysis

To understand the usage of activity timestamps in app data models, I decided to conduct case studies of apps. This approach is targeted to obtain knowledge about the status quo of activity timestamps in apps based on their actual design and programming. Alternatively, I contemplated to conduct experiments with developers, giving them programming tasks and

observing if and where they include activity timestamps. While the latter might be better suited to investigate the underlying causes and motivations of activity timestamp usage, it does not necessarily produce insight into the status quo of real-world apps. Hence, the program analysis of real-world apps and developer experiments should rather be seen as complementary. Such developer experiments have not been part of this dissertation and are left to future work.

Static Usage and Purpose Analysis of Real-World Apps

For conducting program analyses on existing apps, I decided to examine open-source apps due to the possibility to inspect their source code. Source code inspection was presumed necessary, because app documentation is typically not detailed enough to contain information about data model details and in particular timestamps and their purposes. Moreover, only conducting analyses based on the input and output behaviour of an app, e. g., by looking at database schemas or UI content, does not necessarily reveal all purposes, especially if timestamps are only programmatically used but not used in the UI. The following method was applied in an early, exploratory case study of Mattermost (Chapter 3) and for a later evaluation of timestamp alternatives with Taiga (Chapter 5).

Analysis Method Investigating usage and purpose of activity timestamp from source code, has no prior art in the literature that we could find. It requires several intermediate steps to finally be able to deduce what a given data model timestamp is used for. First, an identification of all (personal) timestamp attributes in the data model. Second, a localisation of all uses of these timestamps. And third, a determination and categorisation according to their purpose.

Timestamp Identification Timestamp attributes are identified in the data model by their data type and all finds are manually verified based on their attribute name to control for false positives. The found timestamp attributes are classified as personal data if there exists a simple path of references from the entity containing the timestamp to a user entity.

Usage Localisation Programmatic uses of each identified timestamp attribute are located in a static manner by modifying the type-safe renaming function of a refactoring tool. The modified tool yields every location where the targeted timestamp is accessed, which is then manually examined and classified based on the immediate program context. We performed this analysis on client-server web applications with centralised data storage. As a result of the client-server architecture, programmatic usage can cross the boundary between back-end (server) and front-end (client). Hence, we also traced the respective timestamp identifiers across this boundary and within the client code. We further used manual UI inspection to confirm the visibility of timestamps used for the purpose of informing users.

Purpose Classification Timestamp types and usage purposes are then manually classified following a bottom-up approach considering the attribute naming (type) and local context (purpose). The type of a timestamp hereby refers to a specification of its associated event, e. g., the creation of an object instance, which could be deduced by the descriptive naming of the attribute *createdat*. The usage purpose hereby refers to a functional aspect of the app that is facilitated by the timestamp, e. g., determining new and unseen messages. We found that type and purpose are typically apparent from their context in the source code.

Analysis of Alternatives Based on the determined usage purposes, I contrasted their functional goal with the necessity to use timestamps. To do so, I analysed and described the function that a timestamp fulfils for each usage purpose, e. g., provide a partial order of objects' ages for chronological ordering. Given this function, I described potential substitutes for timestamps that perform the same or similar functionality with reduced privacy impact.

1.3.2 Empirical Data Analysis

Activity timestamp exposure and its related aspects are not well described and understood in the literature so far. To add previously missing evidence, we conducted empirical analyses of public, large-scale and real-word datasets to indicate the user preferences for timestamp precision and the privacy impact of reducing the precision of activity timestamps.

User Preferences for Timestamp Precision

Answering the question of user demand for informative activity timestamps could be approached at least in three ways in terms of methodology: (i) interviewing experts, (ii) evaluate user behaviour, or (iii) deduction from procedural guidelines. Following these approaches would presumably lead to different results, because of the subjectivity of human processes and hidden variables influencing user demand. Ideally, they should be regarded as complementary.

In brief exploratory expert interviews, we soon realised that the quality of participants' answers were heavily limited by their lack of introspection in that matter. Users appear to process informative timestamps subconsciously and find it hard to verbalise their usage purposes. For that reason and because we did not have access to more expert participants, we abandoned the interview as methodology. Similarly, we abandoned the deduction-from-guidelines methodology due to the lack of accessible and sufficiently detailed guidelines.

For evaluating user behaviour, we identified and explored the following empirical approaches to gather evidence about user demand.

Git Configurations While inspecting Git for the purpose of designing a self-protection tool against its timestamp exposure, I noticed its rich configurability in terms of output customisation, including various options to control the presentation of Git dates in its command-line output, and that this customisation is stored in configuration files. These customisation preferences can be seen and interpreted as a user's expression of their underlying demand, in the sense that a deliberately chosen date presentation should arguably cover their expectation of necessary precision information. In other words, the precision conveyed by the customised presentation should be a higher estimate of a user's precision demand.

As no comparable dataset of Git user preferences has been published before, I compiled such a dataset of timestamp precision related customisation features from over 20 000 publicly available Git configuration files on GitHub. The resulting dataset is publicly available [Bur22].

Estimating precision demand from these extracted customisation features requires that the impact of each feature value on Git's timestamp presentation logic is understood and put into deductive rules. To do so, I used manual and automatic documentation analysis and dynamic program analysis to determine Git's behaviour on given configuration and input parameters. These methods were also used to initially discover the relevant features within the configurable options. Based on the determined deductive rules, the conveyed precision can be inferred from each user preference.

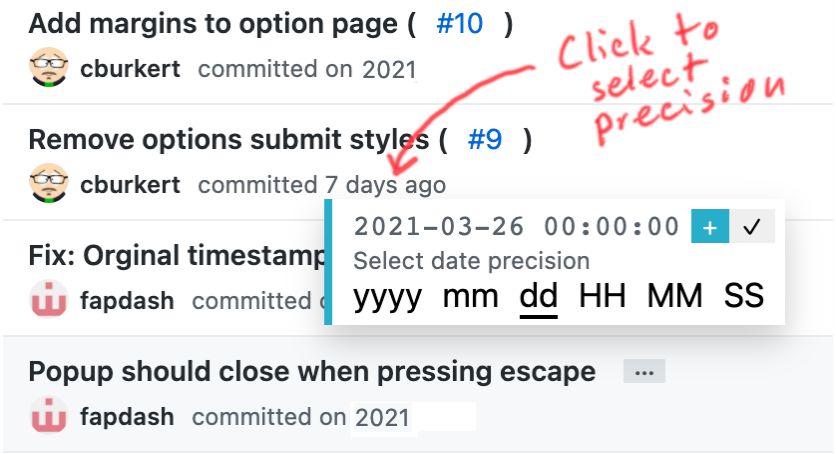


Figure 1.5. Annotated screenshot of the GitHub study's browser web extension showing a pop-up dialogue asking users to select their required level of timestamp precision. All other commits in the shown commit listing have dates reduced to a year precision, while the user increased the date for the commit above the pop-up to day precision.

GitHub User Study GitHub's user interface contains a multiple of Git's activity timestamps, as it offers many additional collaboration features besides hosting and browsing Git repositories. Presumably, the type and frequency of engagement that users have with each feature impacts their respective precision demand. To compare and contrast the precision demand deduced from Git configuration files with the precision demand that users of GitHub might have overall

and for each feature specifically (e.g., issue tracking), we devised and implemented a user study. The study uses a self-developed browser web extension to manipulate timestamps presented on GitHub to reduce or increase the visible timestamp precision. As illustrated in Fig. 1.5, the study first reduces all dates on GitHub's Web UI to a year precision and provides the user with the control to on demand reveal higher precisions individually for each timestamp. By selecting a desired precision level, the user expresses a precision demand which is anonymously recorded by the extension and periodically reported for our study evaluation.

We ran this study setup, with the extension available for download in all major browsers' extension stores, between November 2021 and July 2022 without reaching a significant number of interested participants. Due to the lack of usable results, this approach to determine user demand is not published, and thus not further elaborated in this thesis. The extension is available as open-source software [EMP22b]. Refining this methodology to attract and appeal to sufficient participants remains for future work.

Privacy Impact of Timestamp Precision Reduction

Answering the question what the impact on user privacy of a given timestamp precision reduction level would be, could again be answered through different approaches with respect to the acquisition of data. The acquisition and the subsequent impact evaluation is described in the following.

Data Acquisition I considered a simulative evaluation which uses a descriptive model of various typical workplace activity profiles to generate a global log of simulated user activities. This requires sufficiently detailed descriptive models to be able to generate realistic activity data, including a realistic modelling of the sources of variation, e. g., task difficulty and user performance. Considering the little understand and formalisation of worker processes in collaborative software systems in general and the little process knowledge extractable from documentation and our expert interviews, I decided to not pursue this approach.

Another possible approach is to use already available datasets comprising of timestamped user activity. Large amounts of such activity data can be publicly found on GitHub. Public GitHub data – including user activity data – has been used by the research community for many years (e. g., [Cla+18; MVS21; OHM13; RN16; Ven+13; VFS13]), to the extent that multiple endeavours have been undertaken to form special scientific mirrors of GitHub data that can be more easily accessed by researchers and also take off load from GitHub. We used the GitHub mirror GHTorrent [Gou13]. From this dataset we extracted the metadata of all commits.

Dataset Preparation To ensure that the commit dataset is a fair representation of human commit activity and not significantly skewed by non-human (bot) activity, we first evaluated the distribution of users' overall number of commits made in the time span captured by GHTorrent.

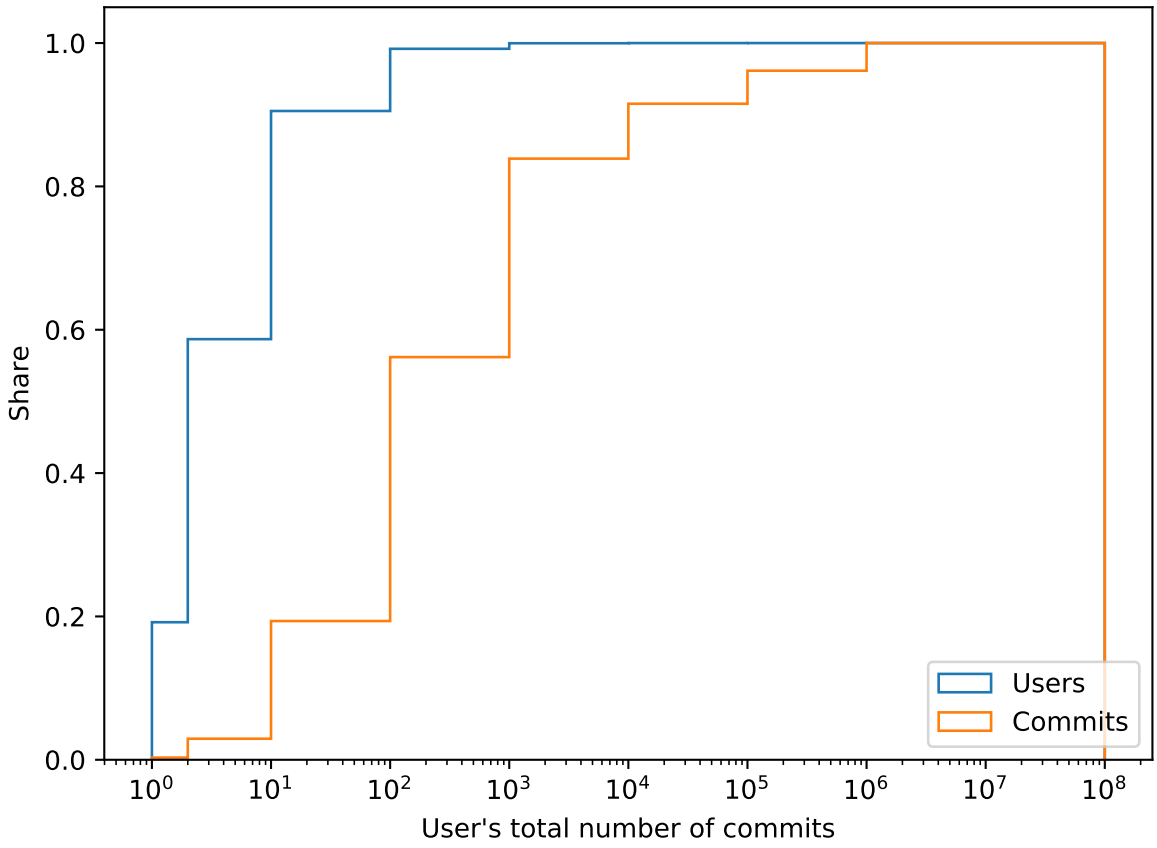


Figure 1.6. Cumulative distribution of users and commits in the GHTorrent dataset as a function of users' total number of commits.

In doing so, likely bots and users with insignificant public engagement on GitHub can be excluded from further analysis.

As depicted in Fig. 1.6 and detailed in Table 1.1, almost 60% of all GitHub users with public code contributions have less than 10 total commits. These users only account for 2.9% of all public commits. The largest share of commits (37%) is contributed by users with between 100 and 1000 commits (8.7% of users), followed by 28% of commits by users with 1000 to 10000 commits (0.78% of users). These figures demonstrate that the selection of an appropriate sub-sample is necessary to avoid a distortion from users with a very low activity, which presumably have overly large interval between activities, and (likely non-human) users with a very high activity, which on the contrary have unrealistically low intervals.

To estimate whether human activity is behind a user's contributions, we calculated the average commit rate that would be required to achieve a given commit total in the period from January 1st 1970, to the date of our dataset download, April 1st, 2019. We picked this early starting date to account for the fact that commits may well predate the introduction of GitHub or Git itself and may originate from much older revision control systems or historic code archives. We argue that this early starting date offers reasonable leeway to account for such phenomenon.

Bin (upper limit)	Users Total	Users Share [%]	Commits Total	Commits Share [%]
2	3 946 605	19.1916	3 946 605	0.3030
10	8 122 024	39.4959	34 331 572	2.6356
100	6 546 395	31.8339	213 863 902	16.4178
1000	1 784 510	8.6777	479 707 748	36.8261
10 000	159 396	0.7751	360 754 696	27.6943
100 000	5018	0.0244	99 758 573	7.6582
1 000 000	247	0.0012	60 028 361	4.6082
100 000 000	19	0.0001	50 239 475	3.8568

Table 1.1. Histogram of commits in the GHTorrent dataset with logarithmic bins of users' total number of commits.

We further assume that only a third of the day is used for productive work, corresponding to a typical eight-hour workday. For simplicity, we do not assume any days off work. The resulting average commit rates would be one every 8 min for a 1 million total, one every 86 min for a 100 000 total, and about every 14.5 hours for a 10 000 total. Considering these rates, we deemed accounts with over 100 000 commits as bot-driven and limited our further analysis to activities within the two groups between 1000 and 100 000 commits.

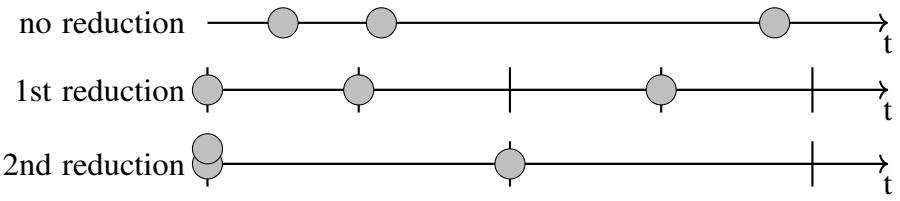


Figure 1.7. Illustration of the collapse of distinct activity points in time as an effect of increasing timestamp precision reduction. In the depicted example, the reduction by removal of higher precision timestamp information leads to no activity times collapsing at the first reduction level. But at the second reduction level, the earliest two activities collapse and appear to coincide as the distinguishing information is lost.

Impact Measurement To estimate the privacy impact of timestamp precision reduction in commit dates, I devised a straightforward metric to assess the ratio of temporally unique activities in a chronological sequence of activities. This assesses how many directly consecutive activities are temporally distinguishable, i. e., for how many activities can an attacker observe an interval, which is information that could be exploited to monitor, e. g., throughput or responsiveness. The ability to temporally distinguish consecutive timestamped activities is of course dependent on the precision of the available timestamp information, and the frequency and distribution with which the activities occur, i. e., the intervals.

Based on that, the experiment to estimate privacy impact uses the described GHTorrent commit information – which incorporates a real-world representation of variability in activity frequency – and applies varying levels of timestamp reduction to the activity timestamps. At each given reduction level, the resulting ratio of temporally unique activities is then calculated per user. The ratio is expected to decrease with increasing precision reduction due to the fact that reduced activity times increasingly loose temporally distinguishing information and thus appear as coinciding events. Figure 1.7 illustrates this effect of activity time collision with an example. By applying this experiment to the commit dataset, we gain evidence about how much decline in distinguishable activities can be achieved at which precision reduction level, which is a direct indicator for privacy impact of precision reduction.

1.3.3 Experimental Security and Privacy Analysis

To investigate whether and how workers might expose their activity times to observers, I conducted experimental analyses of presumably vulnerable work environments. The analyses focus on traffic leakage into untrusted access networks and are explained in the following.

Analysis of Traffic Leakage

The first hypothesis was that mobile workers are exposed to additional risks of activity observation due to the fact that they operate in untrusted and unknown access networks thus opening up their network traffic to network-based observers. As a countermeasure, we assumed that they employ VPN apps to protect traffic confidentiality. The second hypothesis was that VPN apps might fail to ensure a leak-free and reliable operation in such access networks. To test these hypotheses, we used experimental analyses of popular VPN apps on all major platforms. The following describes the text objects, the test bed, and the procedure of the experiment.

Investigated Platforms and Clients We inspected all major platforms for mobile (Android and iOS) as well as desktop/notebook computers (Windows, macOS, GNU/Linux). For these platforms, we included their native, on-board VPN clients as well as popular VPN apps that are targeted as personal VPNs, meaning that their objective is to provide security and privacy protections to their users instead of providing access to a remote private network. Typical features of such personal VPN apps and services are the avoidance of traffic leaks into access networks by routing all possible traffic through the tunnel and ensuring that no traffic bypasses the tunnel, including when the VPN connection might get interrupted. To ensure this functionality, these VPN services come with dedicated VPN apps. These third-party VPN apps, the platforms' native clients, and the platforms' VPN subsystems and APIs on which the apps and clients might built on, each have their own surface for design flaws and implementation errors that can thwart a leak-free VPN operation. Additionally, the functionality to detect and remediate captive networks can be implemented at platform and/or app level and can interfere with VPN operation. To better isolate and understand the influence of each functional

layer, i. e., is an observed behaviour caused by the app or by the platform subsystems, we also implemented a demo VPN app for macOS that directly uses on-board functionality.

Test Bed Setup We built a test bed that emulates a Wi-Fi access network typically found at train stations, hotels, or cafés. The test bed comprises of a Raspberry Pi acting as the Wi-Fi access point, the host for the optionally active captive portal, and the point where we probed the bypassing egress traffic for leaks. The test bed only captured traffic sent from the MAC address of the tested client device that is known to be used for Wi-Fi association from previous connections. Hence, traffic sent form other (randomised) sender addresses like it is common for Wi-Fi probing is not part if this analysis, but subject of other research to which I contributed that is not part of this dissertation [Ans+22]. The test bed was deployed in a controlled lab setup.

Experimental Procedure For each app and each supported platform, we conducted variations of experiments, where we repeatedly associated to the test bed Wi-Fi and observed the egress traffic. The variations included the activation of the captive portal functionality to test whether the respective combination of app and platform can remediate the captive network without leaks and deadlocks. Moreover, we introduced the variation of selectively dropping traffic destined for the respective VPN endpoint to test whether the app handles a failure to establish a tunnel without leaks. In each variation, we observe the traffic emitted in the immediate aftermath (20 sec) of Wi-Fi association, and traffic is regarded as a leak if it is not part of the captive network remediation, VPN establishment or auxiliary lower-level protocols.

1.3.4 Design, Implementation and Evaluation of Privacy- and Transparency-Enhancing Techniques

This dissertation uses constructive methods to provide tooling directed at various stakeholders of the activity timestamp exposure problem. It provides tools directed at data subjects to demonstrate how self-protecting measures can be applied against observability. It provides tools for developers as proof-of-concept for framework and API concepts fostering data minimisation in timestamps. And it provides tools for monitoring and auditing purposes to efficiently compile lists of personal data. Details about these tools are given in Sect. 1.4. The following explains the specifics in their respective methodology.

Tools for Self-Protection

The building of self-protection tools is conditional on the ability to access and interfere with the data processing of the invasive app or system. This interference is necessary to restrict data processing as wanted, but at the same time, the functionality of the interfered-with system should remain intact. Hence, design and evaluation are focused on compatibility which requires an exploration of system interfacing options.

Git Timestamp Exposure As a decentralised version control system, Git determines data locally before later integrating it on shared remote repositories. This local generation gives self-protection approaches the chance to modify data before publication. In practice, Git's integrity protecting hash-chain data structures are adverse to retroactive modifications. To avoid negative effects of creating and publishing diverging hash chains, the design has to explore various methods of interfacing in order to find the least disruptive.

Network Traffic Leakage As discussed earlier, avoiding traffic leakage in public access networks requires a coordinated interplay of operating system, VPN client and all other network-active apps. Particularly the operating system needs to grant and restrict networking capabilities to the involved components in the right order and under the right conditions to avoid leaks and deadlocks. Modern modular operating systems therefore have to provide APIs to let modular service components register and communicate, enabling the operating system to assert its control. For instance, the operating system has to be able to monitor the status of a VPN tunnel to decide if it is safe to grant networking capability to other apps. Modern operating systems like macOS and iOS provide service APIs for VPNs. To evaluate whether these APIs are fit for use and deliver the claimed controls for leak prevention, I conducted API inspections and tests, including the development of a VPN app for macOS to evaluate the API functionality, as port of the experimental security analysis.

Tools for Data Minimisation

Addressing the issue of excessive timestamp usage in app design, we followed a constructive approach to develop and evaluate a framework of more data-minimal timestamp alternatives. For requirement analysis, we examined existing descriptions of timestamp use cases in the literature. In a second iterative design refinement step, we included additional information from a further case study that we conducted for the Taiga app [Tai21a]. The API design was derived and designed based on that information and designed to nudge developers towards deliberate decisions over what the necessary precision is. The framework was evaluated within a case study regarding its implementation effort and cost overhead.

App Data Model Analysis

In looking for a solution for the problem of compiling lists describing the personal data in apps, we followed first an exploratory approach to discover ways of automatically deriving data model graphs from database schemas. In various experiments with real-world database schemas of popular apps, we tested a derivation approach that uses foreign key constraints if they are provided in a schema, and a simple relation mining heuristic otherwise, which we designed and evaluated. Second, we follow an explorative approach to look for a propagation semantic that describes how an entity's property of been related to a data subject affects its related neighbour entities. Third, to associate the determined attributable data with the corresponding data subject

roles, we use graph algorithms to either list attribution paths or check reachability, depending on the acceptable runtime and model size. To add necessary semantic context, a human domain expert is included in the process. We evaluate the resulting process in terms of interaction costs as well as computational and storage complexity.

1.4 Contribution

This section provides an overview of the contributions of my dissertation project included in this thesis. An even more compact summary is provided by Table 1.2. Each of the following subsections bundles the contributions of one paper and groups them according to the pillar they belong to, which is indicated by the prefix *Understand*, *Improve*, or *Maintain*. The subsection correspond to and address the research questions described in Sect. 1.2.

1.4.1 Understand: Timestamp Usage

Understanding how personally identifiable timestamps and activity timestamps in particular are used in app data models is a necessary prerequisite for tackling the privacy issue of activity time exposure. Our literature research did not yield any prior investigation of activity timestamping in data models. Therefore, I claim that we were the first to investigate the issue, describe a methodology to analyse timestamp usage, evaluate it on real-world apps, and propose alternative design patterns.

Method for Timestamp Usage Analysis

I developed and proposed a manual program analysis method to determine personally identifiable timestamps in data models as well as their purposes within the app following a white box approach. By using a white box approach with access to app source code, I was able to investigate all usages within the app back-end and front-end, including timestamps not represented in any app output. I proposed to analyse timestamp types (e. g., creation, modification) as part of the usage analysis and to regard their distribution also in relation to the determined usage types as an indicator for excessive timestamp use. For instance, I could demonstrate in the later case study that deletion timestamps were excessively used for state management. In general, the proposed method is the first approach that produces a classification of timestamp types and usage types, and reveals unused timestamps.

First Case Study of Timestamp Usage

I conducted a case study of the app Mattermost to demonstrate the methodology and gain first evidence about real-word timestamp usage. The case study revealed noticeable patterns in the usage of timestamps with respect to the inclusion of timestamps regardless of purpose. I could

Timestamp Usage and Demand	
Chapter 3	*Towards Minimising Timestamp Usage In Application Software: A Case Study of the Mattermost Application* • Conducted first case study of timestamp usage in apps • Found empirical evidence for excessive timestamp usage • Proposed alternative design patterns to mitigate timestamp use
Chapter 4	*Data Minimisation Potential for Timestamps in Git: An Empirical Analysis of User Configurations* • Gathered first empirical evidence about user demand for timestamp precision • Provided estimation for privacy impact of timestamp precision reduction • Developed tool support to protect against time exposure in Git
Data Minimisation and Mitigation	
Chapter 5	*PrivacyDates: A Framework for More Privacy-Preserving Timestamp Data Types* • Designed more privacy-preserving alternatives for timestamps • Implemented a framework of alternatives for the web framework Django • Integration and evaluation of proposed alternatives with a real-world app
Chapter 6	*Analysing Leakage during VPN Establishment in Public Wi-Fi Networks* • Examine the behaviour and leakage of native and third-party VPN clients • Reviewed the current state of VPN APIs and reported several security issues • Proposed Selective VPN Bypassing to avoid traffic leakage in captive networks
Transparency and Compliance	
Chapter 7	*Compiling Personal Data and Subject Categories from App Data Models* • Describe the problem of ubiquitous identifiability in data models • Propose a semi-automatic expert process to derive personal data and subject categories directly from the source of truth • Provide evidence for its practical applicability and the complexity of real-world data models

Table 1.2. Summary of contributions

observe an excessive inclusion and usage of timestamps, meaning that timestamps were both included without any apparent use and were used for purposes that did not require personal data as privacy sensitive as timestamps. I also observed the phenomenon that only a small subset of all collected and stored timestamps are user-visible, thus increasing the issue of intransparency. The excessive inclusion suggests a presence of questionable design habits that are negatively impacting user privacy. I am the first to suggest the existence of a privacy anti-patterns to excessively include timestamps in data model entities.

Moreover, I contributed to the improvement of open-source software by forwarding my findings and suggestions to Mattermost's product team in August 2019. Unfortunately, they did not get in touch. Most of the findings are still true today.

Timestamp Usage Patterns and Alternatives

I categorised all programmatic uses of timestamps in the case study according to their functional purpose. As a result, I gathered seven types of timestamp usage that include obvious usages like user information but also less user-facing usages like object state management and ETag derivation. For each usage types I discussed the necessity of using timestamps and possible alternatives that allow to fulfil the functional purpose but convey less privacy sensitive information. Therefore, I considered sequence/revision counters, precision reduction, encryption, and enumeration as basis for alternative design patterns.

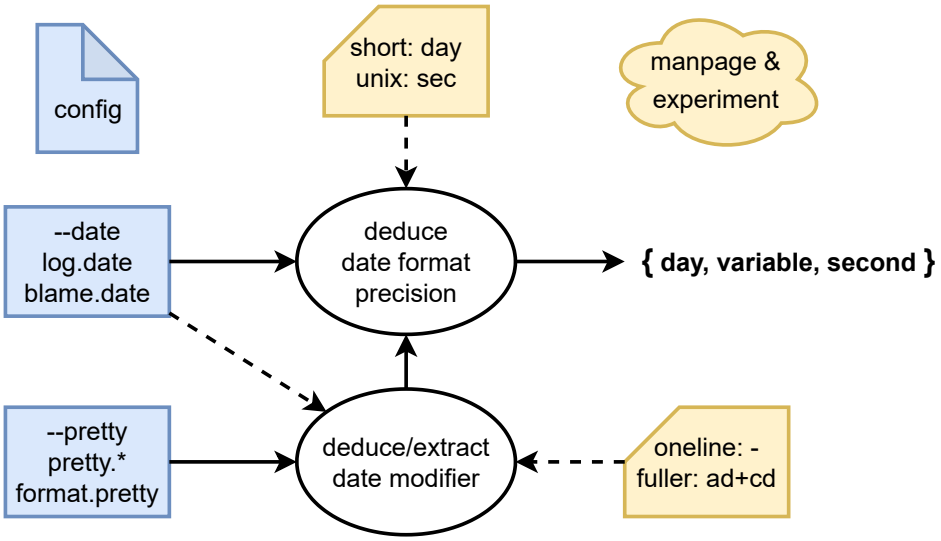


Figure 1.8. Illustration of the deductive process to determine the timestamp precision information implied by a customisation of output presentation by Git commands. The process parses various levels of customisation preferences (`date` and `pretty`) and combines the setting with knowledge extracted from documentation and experiments.

1.4.2 Understand: Timestamp Precision Demand

The status quo is that activity timestamps are recorded as conventional timestamps with second precision or higher regardless of actual precision demand. This behaviour arguably is in violation of basic data protection principles like purpose specification and data minimisation. I have contributed the first evidence to estimate users' precision demand and to estimate the privacy impact of precision reduction. I have further contributed tool support for developers to protect themselves against activity time exposure.

Estimation of Precision Demand

I am the first to investigate user demand for timestamp precision and provide empirical evidence for a more proportionate and therefore more privacy-friendly app design. To do so, I developed an approach to deduce precision demand from user preferences and applied it to Git configuration files. I developed a feature extraction method to identify and extract all preferences related to the processing of Git's activity timestamps in a privacy-preserving way, by eliminating all free-from data. As a result, I have contributed the first public dataset of large-scale user preferences regarding informative activity timestamps [Bur22].

I proposed the estimation of precision demand by analysing user preferences, following the rationale that users do not deliberately choose a configuration with too little timestamp information. Hence, I argued that a user's demand for timestamp precision corresponds to or is below what is achieved by their preferences. In that sense, I claim that utility can be considered as maintained if the level of reduction is below or equal to the determined precision demand.

I have developed a deductive process to reduce user preferences to their inherent timestamp precision, then functioning as a higher estimate of user demand, as explained before. As illustrated in Fig. 1.8, I could deduce a timestamp precision of day, variable and second for timestamp information used in command outputs and a more granular year to second precision range for time-based filters. I could show that about 90% of output customisations did not require a static timestamp with second precision, and that only 0.5% of time-based filter queries use conditions more precise than hours (e. g., '15 min ago'). These figures indicate that conventional timestamps with second precision or higher are not necessary for the vast majority of users and have a high potential for data minimisation.

Estimation of Privacy Impact of Precision Reduction

I have addressed the question whether moderate reductions in precision reduction would even make a significant impact on user privacy. To do so, I have devised a metric that measures the ratio of temporally unique timestamps in a chronologically sorted sequence of activities by a user. To the best of my knowledge, I was the first to propose such a metric to assess timestamp privacy. I used this metric of distinguishable activity times on a real-world dataset of public commits on GitHub and assessed the ratio of distinguishable times depending on the level of timestamp reduction previously applied to the data. In doing so, I am the first to assess the privacy impact of timestamp reduction on real-world data. The results indicate that moderate reductions, e. g., to a precision of one hour reduces the ratio of distinguishable activities to less than half for the median user, while arguably maintaining enough utility, as determined through users' precision demands above.

Self-Protection Tool git-privacy

I have contributed a self-protection tool that can be used by Git users to redact the precision of their Git commit timestamps. The tool is open source [EMP] and provides various features to detect and avoid the accidental publication of unredacted activity times through Git.

1.4.3 Improve: Development of More Privacy-Preserving Timestamp Alternatives

Understanding the potential for data minimisation is only part of the solution to tackle the problem of excessive timestamping. The other part is to provide developers with tools to technically realise those potentials. To the best of my knowledge, we are the first to publish a framework of more data-minimal alternatives to conventional timestamps. We are also the first to implement data minimisation strategies for timestamps and test them on a real-world app.

Design and Implementation of Alternatives

We conducted an analysis of the literature with regard to timestamp data minimisation and related privacy patterns. Based on the found concepts, we derived a consolidated design of alternative timestamp types. The alternative types comprise a type for static precision reduction that nudges developers to select a necessary level of precision and is fully compatible with built-in types, a type for progressive timestamp reduction that allows to reduce timestamp precision in incremental steps based on time elapsed since its creation, and a pseudo-timestamp type in the form of a sequence counter to replace the use cases where timestamps fulfil sorting and ordering purposes but are not used beyond that. We tested and revised our design in a case study where we analysed the timestamp usage of the app Taiga, following a similar approach as with Mattermost earlier. As a result, we proposed an optimised design that combines multiple archetypes of timestamp use, which were identified as commonly co-occurring in the case study, into a single alternative type. We provided with `django-privacydates` an implementation of the design for the popular web application framework Django. The framework is available as open source [EMP22a] and can be used experimentally in Django apps.

Evaluation of the Framework

We evaluated the framework by integrating it in Taiga, where we replaced conventional timestamp types with our alternative types according to their functional purpose. We could demonstrate that the integration required only minor changes depending on the selected type of alternative. Our modified version of Taiga could successfully be operated and showed no negative influences on basic functionality. We further evaluated the storage cost and could show that except for aging timestamp (vanishing dates), all types had similar or even fewer storage costs than conventional timestamps.

1.4.4 Improve: Unobservability of Mobile Workers' Activities

As part of the pillar to improve privacy by minimisation and mitigation, I contributed a novel analysis of traffic leakage vulnerabilities in VPN apps. The analysis regards the common constellation of Wi-Fi access networks that use captive portals to regulate access. In experiments with real-world apps on all major platforms, I could demonstrate that this constellation is endangering user privacy due to insufficiently secured apps and a lack of cooperation between platforms and apps. One common negative effect is the leakage of user traffic into the untrusted access network. The other effect is a deadlock between captive network remediation and VPN establishment that prevents Internet access until users disable their VPN app, which in turn likely leads to traffic leakage as well. To complement the experiments, we analysed platform documentation on VPN APIs. During the experiments, I discovered and reported several bugs and security issues in the tested software. Finally, I proposed a network and VPN bootstrapping process that ensures a leak-free operation by selective VPN bypassing. These contributions are detailed in the following.

Experimental Results

We were the first to describe and investigate the security and privacy risk originating in the interplay of VPN tunnel establishment and captive access networks. We devised an experimental test bed to analyse platforms and VPN apps for occurring traffic leaks. To assess the apps, we identified and described five criteria to avoid leaks in a captive network constellation. We examined multiple app versions of four third-party VPN services, including market leaders, as well as the native VPN clients of the major platforms Windows, macOS, GNU/Linux, iOS, and Android. The examination revealed significant shortcomings: Only one app version was able to prevent both, leaks and deadlocks, while other apps that successfully prevented leaks caused deadlocks by not allowing the captive network remediation to function. We further identified issues with insufficient leak prevention for traffic to platform services as well as race conditions between the setup of leak prevention and the opening of system-wide traffic egress. All in all, our experimental analyses showed that the confidentiality of communication metadata and activity was at risk in almost all cases, despite the use of VPN.

Status Quo of VPN APIs

As the ability for VPN apps to operate securely also depends on the APIs provided by the platform, we examined the status of VPN APIs with respect to their integration depth and feature support. The experimental results suggest that a deep integration is a necessary prerequisite to avoid deadlocks and leaks. The API has to ensure that a VPN startup takes precedence over general networking to ensure that tunnels are set up in advance of other outgoing traffic. Moreover, the API has to establish an understanding between the VPN app and the platform about any traffic suppression undertaken by the VPN app. To that extent, the API has to ensure that vital traffic like captive portal detection is exempt from such suppression. Since not all

VPN apps necessarily aim to provide these functionalities, the API has to be unambiguous about the underlying assumptions and provided guarantees.

We reviewed and analysed the VPN APIs of the tested platforms based on public documentation and found that most of them provide some integration depth and offer certain guarantees like an automatic startup and keep-alive of the interfacing VPN subsystem. We found that Apple's macOS and iOS platforms claim to provide automatic startup and a blocking keep-alive for apps that use the on-demand connection rules. This level of integration depth is more than we could find for the other platforms.

To test Apple's claim, I developed a demo VPN app that uses these API functionalities. By submitting it to the aforementioned experimental analysis, I could demonstrate that the claimed guarantees were in fact not reliable. This previously unreported issue can thwart the security and privacy of VPN apps that also relied on these guarantees. I reported this issue, as detailed in the following section.

Reported Security Issues and Bugs

I have also contributed to the security and privacy of major operating systems by reporting unexpected and faulty behaviour in their VPN components. In the following, I briefly describe the reporting processes and give an update on the current status.

Apple I reported on November 2nd, 2020, to Apple Product Security[8] that their on-demand connection feature of the VPN API did not fully block network connectivity as was described in the API documentation [Appa; Appd]. The feature essentially allows to setup on-demand rules for registered VPN profiles which trigger the automatic establishment of the VPN tunnel if the rule condition matches (e. g., the device connects to a Wi-Fi network). The documentation read: 'When rules of this class match, the VPN connection is started whenever an application running on the system opens a network connection. Network connectivity will be blocked until the VPN is connected.' [Appa]

After numerous inconclusive inquiries about the status of the reported issue, Apple Product Security answered on June 22nd, 2021:

> We have determined that we will not be making any changes to our platforms as a result of your report. Our recommendation is that developers use the NEFilterPacketProvider API in conjunction with the VPN APIs to enforce that traffic routed outside of the VPN is dropped when that behaviour is desired.
>
> We will be updating our documentation to better reflect this resolution.

As of July 13th, 2022, more than a year later, the documentation has not been updated and the incorrect claim is still online as cited above. Developers that rely on the described API behaviour are still at risk of producing VPN apps that leak traffic during connection establishment.

[8] Follow-up reference 751562954

Google Android I reported on October 12th, 2020, to Google's Android issue tracker[9] that their native always-on VPN feature produced deadlocks in captive networks. Google marked the issue as a duplicate of another issue, for which we had no access rights to follow the further discussion. I have since received unverified reports by Android users that the deadlock issue has been resolved.

GNOME I reported on October 9th, 2020, two issues to GNOME's NetworkManager project: a failure to run captive portal detection when a VPN is configured to auto-start on network connection[10] and another unexpected behaviour that prevents automatic startup of a VPN although configured[11]. Both issues were automatically closed after a 6-month absence of reaction to my reporting.

Selective VPN Bypassing Process

Based on the insights from our API documentation and experimental app analysis, we proposed a process to startup network and VPN subsystems in a secure and leak-free way. The proposal describes the order of startup steps and conditions for progressing to the next step or secure fail states. The process clarifies how captive network remediation should be able to bypass VPN tunnel and blocks, and that the establishment of the VPN tunnel should be the prerequisite for the remaining system to gain networking capabilities. Platform maintainers should adhere to this process to avoid the described leak and deadlock issues.

1.4.5 Maintain: Compiling Personal Data from Data Models

We proposed and implemented *Schemalyser*, a semi-automatic expert process to derive listings of personal data from descriptions of app data models. The process uses data model descriptions like database schemes and translates them into a multigraph representation of data model entities and relations between entities corresponding to foreign key relationships. The following describes our contributions for determining identifiability, the attribution to data subject categories, and the tackling of incomprehensible listings. But first, I describe our contribution of describing the ubiquitous identifiability problem which significantly complicates the above.

Problem of Ubiquitous Identifiability

We could demonstrate that the task of identifying all personal data in app data models is more complex than naive approaches might suggest. To the best of our knowledge, we were the first to point out and describe the problem of ubiquitous identifiability. We could show that the high

[9] https://issuetracker.google.com/issues/170461560
[10] https://gitlab.freedesktop.org/NetworkManager/NetworkManager/-/issues/550
[11] https://gitlab.freedesktop.org/NetworkManager/NetworkManager/-/issues/549

degree of cross-linking relationships within real-world app data models has the negative effect, that the vast majority of data model entities become in some form attributable to users through the logic of the data model. In other words, we could show that almost all data is arguably personal data because it is in relation with user activities in the app.

As a result of this effect, a comprehensive listing of personal data is at risk of becoming overwhelming and no longer comprehensible. This problem has not been well researched so far and the explosive growth of attributable data in densely linked data models has not yet been described as a cause and problem for compiling comprehensive and comprehensible records of personal data. To following contributions try to mitigate this issue.

Identifiability of Personal Data in Data Models

We developed graph-based algorithms to markup the identifiability of each data model entity with respect to a given user entity. This markup algorithm introduces a novel distinction in the semantic of identifiability, which reflects the effect that relation directions have on the exclusivity, i. e., cardinality, of identifiable data subjects. The distinction results into two subtypes of identifiability: dedicated and shared, where the former allows to identify an individual data subject through the data model, while the latter identifies a set of individuals. We demonstrated that both types of identifiability are relevant for determining personal data and that a limitation to dedicated data will produce insufficient listings.

Data Subject Categories and Roles

Our contributions include novel approaches to determine and attribute data subject categories as well as a novel role semantic supporting the prioritisation of data.

Determining Data Subject Categories We proposed a method to determine data subject categories by inspecting the relationships that directly surround user entities. We introduced semantic subtypes of these first-hop relations that distinguish whether the thereby connected data is integral to every user of the app, only relevant under certain conditions, or only relevant to users that fulfil a certain role in the app. The role, we argued, corresponds to the concept of data subject categories used in GDPR.

Attribution to Data Subject Categories To provide the necessary transparency about which personal data is processed for which data subject category, we introduced an approach to automatically determine data entities relevant for a data subject category. The approach considers all entities as relevant for which there exists a path in the identifiability markup sub-graph that contains the role-defining graph edge, i. e., if an entity obtains its identifiability through this role relation. We contributed an algorithm that generates such paths on demand for given pairs of entities from storage-efficient identity markup traces. In terms of the practical

usability, we could demonstrate that this algorithm still suffers from a high computational and storage complexity due to the complexity of the underlying general path listing problem, which might render the algorithm unusable for apps with large and dense data models. To mitigate this issue, we also contributed a version of this algorithm that uses the less complex graph reachability for attribution, at the cost of losing the additional interpretation context provided by paths.

Decisive Roles Moreover, we extended the terminology of a data subject category, by introducing the concept of a decisive role. Decisive roles are a subset of all roles to which an entity is attributable. The subset is declared by a domain expert by selecting those roles that are decisive for the value of an attribute, instead of those that are attributable but remain semantically 'passive'. We proposed this novel distinction as a factor to alleviate the problem of ubiquitous identifiability and the resulting problem of comprehensibility for listings of personal data. By introducing decisive roles, we provided a differentiating factor based on the actual semantic relevance that an attribute has for a given data subject category.

Condensed Listing of Personal Data

We introduced a method to create condensed forms of personal data listings. It uses the type of identifiability and the decisive role property as semantic criteria to determine which attributes are likely more relevant for a specific role. All attributes that are thereby determined to be less relevant, are condensed by summarising them with a representative grouping term. More detailed information about grouping terms could be revealed through a nested, i. e., multi-layered, listing of personal data, as suggested by Article 29 Working Party [Art13], meaning that readers of the listing can choose to learn more about the attributes behind specific grouping terms.

Integration into Software Development

We outlined how our process can be used as part of software development to keep track of newly added personal data or changes to relations and the identifiability of entities. We further proposed ways to integrate this mechanism into software development. That way non-inferable semantic knowledge about data models can be added in the form of code annotation directly to the source code, instead of needing to query developers as the domain experts later on. These annotations could declare in a machine-readable way which entities are user entities, which relations define roles, what the dependent roles are for each attribute, and even suggest grouping terms for the condensed listing.

Introduction & Background	Chapter 1 Chapter 2	Introduction Background
Timestamp Usage and Demand	Chapter 3 Chapter 4	Towards Minimising Timestamp Usage In Application Software: A Case Study of the Mattermost Application Data Minimisation Potential for Timestamps in Git: An Empirical Analysis of User Configurations
Data Minimisation and Mitigation	Chapter 5 Chapter 6	PrivacyDates: A Framework for More Privacy-Preserving Timestamp Data Types Analysing Leakage during VPN Establishment in Public Wi-Fi Networks
Transparency and Compliance	Chapter 7	Compiling Personal Data and Subject Categories from App Data Models
Conclusion	Chapter 8 Chapter 9	Thesis Summary Outlook on Future Work

Table 1.3. Tabular outline of this thesis. The included papers (Chapter 3 to Chapter 7) are grouped in three parts corresponding to the three pillars of my dissertation project.

1.5 Thesis Outline

This cumulative thesis comprises a selection of my research papers with relation to the activity timestamp exposure problem. The research papers are grouped according to the main pillars of my work as illustrated earlier in Fig. 1.2. The thesis is concluded by a summary and outlook in Chapter 8 and Chapter 9. To complement this introduction, Chapter 2 first provides additional background on the topics that could not be included in the papers. The outline is summarized in Table 1.3.

The remainder of this section explains the grouping of my papers into the three pillar parts.

Timestamp Usage and Demand This part of my thesis includes my papers that mainly contributed towards a better understanding of both timestamp usage in apps (Chapter 3) and users' demands for timestamp precision (Chapter 4). Both contributions lay the groundwork for data minimisation by providing methodologies and empirical evidence to inform more data minimal app designs.

Data Minimisation and Mitigation Building on the understanding of timestamping and the resulting exposure, this part includes two papers that contribute to the mitigation of activity timestamp exposure. Chapter 5 mitigates the sensitivity of exposed timestamp information by designing and implementing data minimisation techniques for data model timestamps. Chapter 6 complements the mitigation efforts on collected data by looking at activity exposure

via network traffic. It proposes approaches to better mitigate network exposure by hardening VPN apps against a common vulnerability that we could demonstrate for users in public Wi-Fi networks.

Transparency and Compliance This third part of my thesis concerns approaches to facilitate the compliance of operating apps and increasing transparency for data subjects as well as developers and auditors. Chapter 7 introduces a new process to compile personal data and data subject categories based on information in app data models and additional context from domain experts.

1.6 List of Publications

The following is a list of the publications included in this cumulative dissertation.

1. Christian Burkert and Hannes Federrath. *"Towards Minimising Timestamp Usage In Application Software: A Case Study of the Mattermost Application"*. In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Ed. by Cristina Pérez-Solà, Guillermo Navarro-Arribas, Alex Biryukov and Joaquin Garcia-Alfaro. Vol. 11737. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 138–155. ISBN: 978-3-030-31500-9. DOI: 10.1007/978-3-030-31500-9_9
2. Christian Burkert, Maximilian Blochberger and Hannes Federrath. *"Compiling Personal Data and Subject Categories from App Data Models"*. In: *ICT Systems Security and Privacy Protection*. Ed. by Audun Jøsang, Lynn Futcher and Janne Hagen. Vol. 625. IFIP Advances in Information and Communication Technology. Cham: Springer International Publishing, 2021, pp. 242–255. ISBN: 978-3-030-78120-0. DOI: 10.1007/978-3-030-78120-0_16
3. Christian Burkert, Johanna Ansohn McDougall, Hannes Federrath and Mathias Fischer. *"Analysing Leakage during VPN Establishment in Public Wi-Fi Networks"*. In: *ICC 2021 - IEEE International Conference on Communications*. ICC 2021 - IEEE International Conference on Communications. Montreal, QC, Canada: IEEE, June 2021, pp. 1–6. ISBN: 978-1-72817-122-7. DOI: 10.1109/ICC42927.2021.9500375
4. Christian Burkert, Jonathan Balack and Hannes Federrath. *"PrivacyDates: A Framework for More Privacy-Preserving Timestamp Data Types"*. In: Sicherheit 2022: Sicherheit, Schutz und Zuverlässigkeit: Konferenzband der 11. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI), 5.-8. April 2022 in Karlsruhe. Ed. by Christian Wressnegger, Delphine Reinhardt, Thomas Barber, Bernhard C. Witt, Daniel Christopher Arp and Zoltán Ádám Mann. GI-Edition. Proceedings Volume P-323. Gesellschaft für Informatik e.V. (GI), 2022. ISBN: 978-3-88579-717-3

5. Christian Burkert, Johanna Ansohn McDougall and Hannes Federrath. *"Data Minimisation Potential for Timestamps in Git: An Empirical Analysis of User Configurations"*. In: *ICT Systems Security and Privacy Protection*. Ed. by Weizhi Meng, Simone Fischer-Hübner and Christian D. Jensen. IFIP Advances in Information and Communication Technology. Cham: Springer International Publishing, 2022, pp. 323–339. ISBN: 978-3-031-06975-8. DOI: 10.1007/978-3-031-06975-8_19

I also contributed to the following research papers that are not included in this thesis:

1. Matthias Marx, Erik Sy, Christian Burkert and Hannes Federrath. *"Anonymity Online – Current Solutions and Challenges"*. In: *Privacy and Identity Management. The Smart Revolution: 12th IFIP WG 9.2, 9.5, 9.6/11.7, 11.6/SIG 9.2.2 International Summer School, Ispra, Italy, September 4-8, 2017, Revised Selected Papers*. Ed. by Marit Hansen, Eleni Kosta, Igor Nai-Fovino and Simone Fischer-Hübner. Cham: Springer International Publishing, 2018, pp. 38–55. ISBN: 978-3-319-92925-5. DOI: 10.1007/978-3-319-92925-5_4

2. Matthias Marx, Maximilian Blochberger, Christian Burkert, Dominik Herrmann and Hannes Federrath. *"Privatsphäre als inhärente Eigenschaft eines Kommunikationsnetzes am Beispiel einer Anonymisierungslösung für IPv6"*. In: *Die Fortentwicklung des Datenschutzes: Zwischen Systemgestaltung und Selbstregulierung*. Ed. by Alexander Roßnagel, Michael Friedewald and Marit Hansen. DuD-Fachbeiträge. Wiesbaden: Springer Fachmedien, 2018, pp. 209–223. ISBN: 978-3-658-23727-1. DOI: 10.1007/978-3-658-23727-1_12

3. Erik Sy, Christian Burkert, Hannes Federrath and Mathias Fischer. *"Tracking Users across the Web via TLS Session Resumption"*. In: *Proceedings of the 34th Annual Computer Security Applications Conference*. ACSAC '18. New York, NY, USA: Association for Computing Machinery, 3rd Dec. 2018, pp. 289–299. ISBN: 978-1-4503-6569-7. DOI: 10.1145/3274694.3274708 *Received Outstanding Paper Award*

4. Erik Sy, Christian Burkert, Hannes Federrath and Mathias Fischer. *"A QUIC Look at Web Tracking"*. In: *Proceedings on Privacy Enhancing Technologies* 2019.3 (1st July 2019), pp. 255–266. ISSN: 2299-0984. DOI: 10.2478/popets-2019-0046

5. Erik Sy, Christian Burkert, Tobias Mueller, Hannes Federrath and Mathias Fischer. *"QUICker Connection Establishment with Out-Of-Band Validation Tokens"*. In: *2019 IEEE 44th Conference on Local Computer Networks (LCN)*. 2019 IEEE 44th Conference on Local Computer Networks (LCN). Oct. 2019, pp. 105–108. DOI: 10.1109/LCN44214.2019.8990785

6. Maximilian Blochberger, Jakob Rieck, Christian Burkert, Tobias Mueller and Hannes Federrath. *"State of the Sandbox: Investigating macOS Application Security"*. In: *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*. WPES'19. New York, NY, USA: Association for Computing Machinery, 11th Nov. 2019, pp. 150–161. ISBN: 978-1-4503-6830-8. DOI: 10.1145/3338498.3358654

7. Ephraim Zimmer, Christian Burkert, Tom Petersen and Hannes Federrath. *"PEEPLL: Privacy-Enhanced Event Pseudonymisation with Limited Linkability"*. In: *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. New York, NY, USA: Association for Computing Machinery, 30th Mar. 2020, pp. 1308–1311. ISBN: 978-1-4503-6866-7

8. Erik Sy, Tobias Mueller, Christian Burkert, Hannes Federrath and Mathias Fischer. *"Enhanced Performance and Privacy for TLS over TCP Fast Open"*. In: *Proceedings on Privacy Enhancing Technologies* 2020.2 (1st Apr. 2020), pp. 271–287. DOI: 10.2478/popets-2020-0027

9. Ephraim Zimmer, Christian Burkert and Hannes Federrath. *"Insiders Dissected: New Foundations and a Systematisation of the Research on Insiders"*. In: *Digital Threats: Research and Practice (DTRAP)* (22nd Oct. 2021). DOI: 10.1145/3473674

10. Johanna Ansohn McDougall, Christian Burkert, Daniel Demmler, Monina Schwarz, Vincent Hubbe and Hannes Federrath. *"Probing for Passwords – Privacy Implications of SSIDs in Probe Requests"*. In: *Applied Cryptography and Network Security*. Ed. by Giuseppe Ateniese and Daniele Venturi. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022, pp. 376–395. ISBN: 978-3-031-09234-3. DOI: 10.1007/978-3-031-09234-3_19

2 Background

In this chapter, I do not want to repeat the background already provided in the following chapters but go into the more legal aspects behind my work that always had to be kept short within the papers. In many ways, the activity time exposure issue is affected by data protection regulation and its interpretation by the stakeholders. GDPR's aim to be technology neutral creates a wide scope of interpretation on aspects like the obligation to inform data subjects about which data is collected, the required specificity of declared information, or the limits of the obligation to implement data minimisation.

In the following, I try to sketch the prevailing views from legal commentaries and publications on the legal footing of my work. At times, I will allow myself to add my own commentary where I think that current views are at odds with technical realities.

2.1 Obligation to Inform Data Subjects

This section investigates to what extent controllers that operate software are obliged by GDPR to inform users about which personal data is collected by the app. The extent of such an obligation has a direct effect on how well users can inform themselves about the app and make informed decision. It also has a direct effect on the effort controllers need to make in compiling such information.

2.1.1 Information Obligations in GDPR

GDPR distinguishes different kinds of information obligations depending on whether the personal data was collected from the data subject directly (Article 13) or from a third party (Article 14). Both articles provide a different list of information that is mandatory to provide: While Article 14(1)(d) explicitly lists 'the categories of personal data concerned', there is no equivalent in Article 13. This omission might be due to the imagination that the data subject will necessarily become aware of the categories of personal data if they themselves provide them. Taeger and Gabel [TG22, Art. 14 Rn. 8] follow that logic and state that, other than with Article 13, data subjects might in cases of Article 14 not know which data is concerned. Similarly, Kuner et al. [Kun+20] stipulate that Article 14 requires the additional information to compensate for the lack of transparency that results from the collection at a third-party source. The same reasoning is given by the Article 29 Working Party [Art18, p. 36]. However, for this reasoning to hold true, the wording 'collected from the data subject' in Article 13 would

require a narrow interpretation that expects an active involvement of the data subject to the extent that they gain awareness of all collected data, e. g., as if they filled in a form.

2.1.2 Difference in Applicability between Article 13 and 14

According to Taeger and Gabel [TG22, Art. 13 Rn. 6], Article 13 is applicable if the collection happens with the knowledge or the participation of the data subject. Seemingly contradictory, Simitis et al. [SHS19, Art. 13 Rn. 6] state that Article 13 does not exclude collections without the knowledge and participation of the data subject. But Taeger and Gabel [TG22, Art. 13 Rn. 6] also state that the awareness of the data subject is not significant. This is also supported by Kühling and Buchner [KB20, Art. 13 Rn. 13], who state that the awareness of the data subject about the data collection is 'irrelevant'. They say that Article 13 applies if the data subject serves as the direct source of data, e. g., by having their behaviour synchronously perceivable by the controller. Moreover, the Article 29 Working Party clarifies that Article 13 applies also to automatic collections regardless of the data subject's consciousness of the process [Art18, Para. 26]. All in all, the prevailing interpretation appears to not consider the data subject's awareness a necessity for Article 13's applicability.

2.1.3 Implication on Software-based Data Collection

If user awareness is not necessary, then all personal user data collected by software, obvious or hidden, could be considered directly collected from the data subject. For instance, an app that collects and records a timestamp with each user interaction but without the user seeing the timestamp, would also meet this wide interpretation. But such a wide interpretation would contradict the implicit assumption in Article 13 that the data subject has, through their direct involvement, more transparency about the processing than in cases of Article 14.

Consequentially, for Article 13 not requiring to inform about the collected categories of personal data seems like a significant oversight and a misunderstanding of software-based data processing. Following the prevalent legal commentary, in not making the data subject's awareness a prerequisite of Article 13's applicability, effectively thwarts the intended transparency and might put the data subject in a worse position compared to third-party collection regulated by Article 14.

Considering instead that Article 13 would not apply due to a narrow interpretation and a requirement of user awareness, then Article 14 would be the complementary rule that applies if 'personal data have not been obtained from the data subject'. As stated before, it mandates the information about the categories of personal data, which would mean that the controller had to provide data subjects with a list of categories of personal data that are collected from other sources or beyond the awareness of the user.

2.2 Specificity of Declared Purposes and Data Categories

For previous data protection legislation, the lack of clear criteria about the required specificity of declarations was seen as a weak point for privacy. Expectations that GDPR would provide clearer guidelines were not met. Dammann [Dam16] criticises that GDPR failed to clarify what level of abstraction is sufficient when specifying purposes, especially when it comes to assessing compatible purposes.

Creating declarations with higher specificity and detail will likely cause the controller more effort, not only during creation but also when maintaining the information. Therefore, it can be expected that controllers are inclined towards less specificity. Reidenberg et al. [Rei+16] observed this lack of specificity in privacy policies through the use of ambiguous wording. Naturally, the question is: where to draw the line between sufficiently specific and overly vague declarations?

The following covers how legal commentaries discuss this question. The issue of purpose specificity and the specificity of categories of personal data are considered separately, as commentaries usually consider them separately as well. The issue of purpose specification receives much more attention by the literature, however, many of its considerations are transferable to the issue of category specificity.

2.2.1 Declaration of Purposes

The legal view on necessary purpose specificity has direct consequences on my work. The ability to judge the appropriateness of the precision of informative timestamps is limited by the specificity of their declared purpose. Because if purposes are declared rather vaguely then necessary information values can only be deduced with high uncertainty. Moreover, software manufacturers' incentive to monitor the programmatic uses of their apps will arguably be much lower if legal authorities deem generic umbrella terms for purposes sufficient.

Function and Criteria for Specificity

Kuner et al. [Kun+20] state with respect to purpose limitation that purposes must be 'unambiguous and clearly expressed'. Kühling and Buchner [KB20, Art. 5 Rn. 35] stipulate that purposes need to be specified in a way that makes their limitation of the processing transparent to data subjects as well as auditors. Taeger and Gabel [TG22, Art. 5 Rn. 24] claim that the required level of specificity is case specific and dependent on factors like the number of data subjects, the geographical region affected by the processing, and usual extent within a processing context. The commentary is, however, silent on their concrete consideration.

Simitis et al. say that a purpose should be general enough to allow flexibility in data processing but specific enough to still protect the right to privacy and to prevent a loss of context or widening of the usage [SHS19, Art. 5 Rn. 71]. They further state that the purpose must be

expressed as clearly as needed to avoid any ambiguity regarding the intended meaning of the specified purpose [SHS19, Art. 5 Rn. 76]. An overly generic purpose would thwart its function to provide transparency to the data subject [SHS19, Art. 5 Rn. 78], or to meaningfully assess its legitimacy [SHS19, Art. 5 Rn. 79]. Also, the principle of data minimisation could not be fulfilled if purposes were specified in an abstract and generic manner [SHS19, Art. 5 Rn. 80]. They conclude that all points to an interpretation that demands a narrow, precise and concrete specification of purpose to fulfil the various functions of data protection [SHS19, Art. 5 Rn. 86].

The Article 29 Working Party published an opinion on purpose limitation regarding EU's data protection directive that proceeded GDPR.[1] They state in Annex 3 [Art13, pp. 51-55] that specifying purposes has to consider the context and its ambiguities, e. g., if the processing goes 'beyond what is customary'. They make clear that purposes should be more specific where ambiguities arise. As examples they list an application process where it is obvious and customary that a CV was used for assessment of qualification but not if it was used for 'promotion exercises'. However, as they mostly do not provide explicit purpose wordings, it remains unclear how ambiguity is the underlying issue and not rather the withholding of a second purpose.[2]

Further examples by Simitis et al. state that purposes like 'ensure customer satisfaction' or 'providing individual services' are not sufficiently specific [SHS19, Art. 5 Rn. 88]. For a sufficiently specific purpose they provide the example 'defence against a possible claim for damages due to the disputed defectiveness of the kitchen cabinet delivered on 16.8.2018' [SHS19, Art. 5 Rn. 89]. The latter example leads to the impression that, due to the mentioned good and date, the authors expect a sufficiently specific purpose to uniquely identify a data processing event or occurrence rather than being a template for repeated data processing events or occurrences that all follow the same procedure.

Misleading Notion of Specificity

Providing purposes at a level of specificity that identifies individual processing occurrences is logically not possible for purposes that are listed in the record of processing activities (Article 30). Here, purposes have to be templates of repeated processing occurrences as they have to describe the processing detached from identifiable occurrences. The specificity of a purpose can thus not depend on the explicit identification of a single processing occurrence. An adopted version of the example that is usable for records of processing activities could read 'defence against a possible claim for damages due to the disputed defectiveness of a delivered good'.

[1] A similar opinion paper on purpose limitation in GDPR is not available to date. However, their concepts of purpose specification are considered to be essentially identical.

[2] This distinction is missing throughout the WP29 examples. If processing de facto comprises purposes A and B but the expressed purpose only suggests A, then this is not an issue of specificity or ambiguity but of completeness or correctness.

This distinction of specificity of *action* versus specificity of *instance*, is particularly relevant for defining purposes in software systems, where data processing procedures are designed to be repeated. Requiring a purpose to be as specific as 'evaluate the authenticity of your login attempt on June 27th, 2022, at 9:14' would entail a fundamentally different software and user interaction design than without the specification of date and time. Most notably, users would have to be informed every time they logged in, instead of on first usage, because the individual occurrence would constitute a new purpose. This would cause a level of disruption that is hardly compatible with already high degrees of interaction fatigue.

Influence of Processing Complexity

In another example, the Article 29 Working Party contrasts a small fashion boutique store and an online fashion store. The latter, they say, would have to be more detailed and comprehensive in their purpose specification, if they employed analytics and personalised offers. While it stands to reason that a complex processing mandates a purpose description of proportionate extent, it is unclear why a less complex processing could be sufficiently described by a less detailed and comprehensive purpose. If purposes are expected to fulfil the functions of transparency and accountability, a data subject or an auditor cannot safely assume that a lack of detail in a purpose specification is a result of the apparent simplicity of the processing. A lack of detail might just as well hide decisive information like it could for a process that is apparently more complex.

Influence of Common Understanding

Some argue that purposes that fall within a category of common understanding can be expresses in less detail, assuming that the data subject has a prior knowledge about what the processing entails. The Article 29 Working Party (WP29) provides the example of a policy statement by a retail bank that includes the purposes 'provide the financial services' and 'prevent fraud and abuse of the financial system' [Art13, p. 53]. They argue that the first was 'clear and precise enough for most clients to understand the basic scope of the processing', whereas the second was 'too general to serve as a useful specification of purpose'. This distinction appears, however, vague and one could also argue the opposite in the sense that 'providing a service' potentially entails such a range of sub-proposes like credit rating or risk assessment that the scope is hardly made clear at this abstraction.

The WP29's reasoning implies the conjecture that the inner workings of financial services are somewhat understood by a sufficiently high number of potential data subjects. Such conjectures are naturally prone to misconceptions by the information recipient and therefore probably better avoided.

In general, data subjects cannot be expected to gain transparency about a processing based on their assumed preconceptions, which might easily be outdated or naive. The WP29 opinion itself states earlier that purposes should be 'clear to all involved, irrespective of their different

cultural/linguistic backgrounds, level of understanding or special needs' [Art13, p. 17]. Considering that, determining the specificity of a purpose should *not* be based on conjectures about the common understanding of a processing context.

Compromise of Layered Purposes

The WP29 opinion suggests using a layered specification to provide increasing layers of detail ranging from the essential level of detail to a level of detail which provides more explanation or background knowledge [Art13, p. 16]. They say that this balances being 'very concise and user-friendly' with potentially required 'further clarification'. This suggestion is reiterated in a later guideline on transparency [Art18, Para. 35f]. I also proposed a layered-presentation compilation of personal data in Chapter 7.

Implications

The prevailing view among the reviewed legal commentaries and guidelines is that purposes have to be sufficiently specific to provide transparency. It is, however, evident that the interpretation of sufficient specificity is immature and sometimes contradictory even within the same text. Some authors argue for a relative determination of required specificity but fail to appreciate the negative impact on transparency and equal comprehensibility. Moreover, some presented examples indicate a lack of understanding in which degrees of freedom the specificity can be meaningfully adjusted.

For software manufacturers, the reviewed commentary and guidelines provide little actionable information. They are certainly well advised to employ a layered presentation of purpose information. Regarding, guidelines for specifying purposes, legal commentators, data protection authorities and researchers need to consider more use cases of real-word complexity to derive non-trivial and mature exegeses.

2.2.2 Categories of Personal Data and Data Subjects

As part of specifying the minimum content of compliance information, GDPR uses the terms 'category of data subjects' and 'category of personal data'. GDPR does, however, not specify what a category means and how much detail a controller needs to provide in order to fulfil their obligations. The interpretation of the necessary detail has wide-ranging implications on the effective transparency and control that data subjects have about a processing. The following reviews legal commentaries and agency guidelines for clarifications on how to provide such categories.

Function and Required Specificity

The obligation to list categories of data subjects and personal data features in Article 14, where they are part of the information to data subjects, and in Article 30, where they are part of the records of processing activities. Although the addressee of the information is different in both cases, its function is similarly to provide transparency and accountability.

The function of a record of processing activity is to provide transparency and to list all information that is necessary for auditors [TG22, Art. 30 Rn. 7]. Taeger and Gabel [TG22, Art. 30 Rn. 14] state that the used categories should ensure a differentiation to other categories and not be an umbrella term that could effectively enclose all persons. Similarly, Kühling and Buchner [KB20, Art. 30 Rn. 19] stipulate that the level of detail in specifying categories should enable auditors to assess the processing.

With respect to categories in notices to data subjects, Kuner et al. [Kun+20] point out that some data categories might be 'too wide' to 'ensure fairness', i. e., provide meaningful transparency to data subjects. As an example, they say that 'data related to health' might be too vague and instead, 'blood pressure' could be more appropriate. Taeger and Gabel [TG22, Art. 14 Rn. 8] see it as sufficient to provide 'general' information about the processed data that enables a data subject to estimate the associated risks.

Simitis et al. [SHS19, Art. 30 Rn. 24] state that for companies, categories of data subjects like 'employees', 'customers' and 'third parties' might usually be sufficient. Whether such wide umbrella terms are sufficient depends on whether they are suitable to assess the legitimacy of the processing. The commentary mentions that data processing with higher risks for the data subject would demand more detailed categories. For categories of personal data, they suggest terms like 'contact data' or 'profile data', but again caution that more specific descriptions might be needed for processing with higher risk [SHS19, Art. 30 Rn. 25].

The Conference of German Data Protection Agencies (DSK) issued notes for the application of Article 30 in which they detailed the obligation to list categories of data subjects and personal data by providing examples like 'performance data' for employees [Dat18, p. 5f]. They are, however, silent on explicit guidelines for the required specificity of categories.

Mapping Between Data Subject and Personal Data Categories

GDPR does not explicitly demand that a category of personal data should be assigned to the category or categories of data subjects for which they are processed. However, this obligation is seen as implicit by Taeger and Gabel [TG22, Art. 30 Rn. 14]. Otherwise, readers had no way of determining whether a given category of data is appropriate for its context or might even get the incorrect impression that data is collected for one group of data subjects while it is in fact collected for the other. Other commentaries are silent on that matter. Obtaining this mapping is part of our method described in Chapter 7.

Article 15 as an Instrument for More Detail

Simitis et al. [SHS19, Art. 14 Rn. 5] declare that the given categories should be abstract descriptions (e. g., credit rating) that enable a data subject to take further action like an Article 15 inquiry. Also, Taeger and Gabel [TG22, Art. 14 Rn. 8] state that data subjects can use requests according to Article 15 to get more detailed information about their processed data.

It should be noted that data categories used in Article 14 or Article 30 fulfil a fundamentally different function compared to data in Article 15. The two are in a concept-instance relationship, where the former are described as concepts (data categories) and the latter as instances of the concept (concrete data). Hence, it is important that the demand for more or less detail in category description is not confused with the distinction between concept and instance representations. Moreover, category descriptions need to cover the whole extent of the data processing, while instance data naturally comprise only the data that an individual produced. In fact, for instance data to comprehensively represent the totality of a data processing activity, the instance has to cover all potential cases of the processing, which is not generally given. Consequently, the right to query instance data through Article 15 cannot compensate the lack of specificity in conceptual data category descriptions.

Implications

Commentary and guidelines seem to agree that the specificity in category descriptions needs to be sufficient for data subjects and auditors to assess the processing. However, the views on what constitutes sufficient specificity seem to diverge. Most given examples suggest a level of abstraction that is well above what could allow an assessment of data minimisation. It arguably adds no transparency to specify that a shop software collects ‘order information’. Categories at a specificity like this can hold the controller accountable only for the grossest of violations, e. g., if the shop system also collected ‘customer sexual orientation’, which seems unlikely. But if the shop summarized this under ‘customer data’, no one would suspect it either. As a result, listings of categories of personal data and data subjects need to be more specific to provide meaningful transparency, e. g., at the level of ‘blood pressure’ instead of ‘health data’.

2.3 Limits of Data Minimisation

Much of my work aims to either directly provide data minimisation techniques (Chapter 5) or methods to inform data minimisation choices (Chapters 3 and 4). Despite these contributions to facilitate data minimisation, it is still an effort that controllers might rather want to avoid if they are not strictly obligated. This section therefore investigates the legal view on the limits of the data minimisation obligation, if there are any.

While Article 5 does not set any conditions that limit obligations like data minimisation, Article 25(1) does. The latter regulates that and to which extent the controller has to take

technical and organisational measures to implement the principles from Article 5. Here, the obligation to implement measures is relativised as far as the following shall be 'taken into account': (i) the state of the art, (ii) the cost of implementation and (iii) the nature, scope, context and purposes of processing as well as (iv) the risks of varying likelihood and severity for rights and freedoms of natural persons posed by the processing.

Here, the lawmaker clearly intended to codify trade-offs to avoid undue technical burdens, but also to codify criteria to determine the minimum effort. However, this now raises the question how these factors should be 'taken into account'. Could a plea of 'undue implementation cost' excuse a controller from observing the data minimisation principle and thus allow them to process more personal data than necessary? Essentially, how much discretionary power is given to the controller through the consideration of the enumerated factors?

Considering, as a motivating example, a standard software solution to collaboratively edit a spreadsheet document. The controller needs the real-time collaboration feature but does not make use of the detailed document history which lists every edit since the creation of the document. This additional and unnecessary processing of personal data by the history feature would clearly be in violation of the data minimisation principle. If we assume that the history feature cannot be deactivated in the software, could the controller still legally operate the software if they argued that it is state of the art in this type of software and that a custom modification would have undue implementation cost?

The following details the interpretation of legal commentary and guidelines on the limits of obligations by Article 25.

2.3.1 Consideration of the Factors in Article 25(1)

The European Data Protection Board clarifies that the four listed 'elements' in Article 25(1) should be seen as 'factors to be considered together to reach the objective'. They 'contribute to determine whether a measure is appropriate to effectively implement the principles.' [Eur20, Para. 17]. How guidelines and commentary envision the consideration of each factor is reviewed in this section.

State of the Art The European Data Protection Board states that 'the current progress in technology that is available in the market' has to be taken into account [Eur20, Para. 19]. They suggest that the existence and recognition of, e. g., frameworks, 'may play a role in indicating the current state of the art' [Eur20, Para. 22]. Taeger and Gabel [TG22, Art. 25 Rn. 55] explain that to consider the state of the art means that only measures need to be taken that are proven, available on the market, and common practice. Similarly, Simitis et al. [SHS19, Art. 32 Rn. 25] say that there is no requirement to implement 'the newest academic trends and prototypes'. Kühling and Buchner [KB20, Art. 25 Rn. 21] also state that measures need to be sufficiently available in practice to be considered as state of the art, and that this is to be considered in support of the controller not needing to implement measures beyond that.

These interpretations suggest that, for measures to be considered beyond state of the art, they do not have to be technically complex or new, but just have to lack market adoption for arbitrary reasons. Assuming that controllers are not keen to adopt protective measures, this could create an effect of collective non-adoption for the sake of keeping a conservative state of the art. Hence, if one criterion for state of the art is market adoption, the market has discretion about the state of the art and consequently about the appropriateness of measures.

Cost of Implementation The cost aspect cannot be used as an argument to not sufficiently implement measures [Eur20, Para. 24f]. Similarly, Simitis et al. [SHS19, Art. 32 Rn. 26] clarify that lack of financial or personal means is no justification for not undertaking necessary measures. Taeger and Gabel [TG22, Art. 25 Rn. 60] state that there is no limitation of what is reasonable or appropriate based on implementation cost. Instead, cost is to be taken into account to judge, if a potential reduction in risk is still proportionate. Moreover, the cost is not determined on an individual basis, i. e., for an individual controller, but from a general perspective [TG22, Art. 25 Rn. 60].

Nature, Scope, Context and Purposes of Processing The European Data Protection Board [Eur20, Para. 26f] gives an exegesis of the used terms but is silent on how they should be considered. Simitis et al. [SHS19, Art. 32 Rn. 27] provide examples that the processing of particularly large amounts of data or the processing of sensitive data might impact the selection of appropriate measures due to their higher risk. Kühling and Buchner [KB20, Art. 25 Rn. 20] conclude that the requirements for the implemented measures will likely increase for more 'intensive or risky' processing.

Risks Posed by the Processing The European Data Protection Board [Eur20, Para. 30] explains that the controller has to identify the risks presented by a violation of a principle like data minimisation and implement measures to effectively mitigate them. The board stipulates that 'the risk-based approach does not exclude the use of baselines, best practices and standards' [Eur20, Para. 32].

2.3.2 Extent of a Controller's Discretionary Power

Kühling and Buchner [KB20, Art. 25 Rn. 19] conclude that the obligations to implement measures leave a considerable margin of discretion for the controller that can hardly be audited by data protection authorities, safe for gross violations. They stipulate that Article 25 does not set absolute requirements for the implementation of measures, but only requires measures that are proportionate [KB20, Art. 25 Rn. 22].

Taeger and Gabel [TG22, Art. 25 Rn. 47] also state that GDPR does not demand disproportionate measures, however, they emphasise that it does not leave the protection of data subjects' rights to the disposition of the controller. They continue to say that the scope of the controller's discretion

is about which type and extent of 'flanking measure' to employ case by case. Furthermore, they state that controllers do not have to take the best among risk-adequate measures [TG22, Art. 25 Rn. 52].

Simitis et al. [SHS19, Art. 25 Rn. 37] stipulate that it would be wrong if the controller used the factors like cost as excuses to not implement necessary measures. They say that the consideration of the factors defines lower and upper boundaries for the measures that should be taken.

This interpretation would mean that controllers have no discretion about taking adequate measures but are only free to select among adequate measures. Logically, this would mean that measures could be similarly risk adequate yet somehow qualitatively different. But if a measure is better at mitigating a risk than another measure then they can only be considered as similarly adequate if the difference in remaining risk is ignored. If it is in their discretionary power to ignore this difference in remaining risk, then controllers effectively have, to some extent, the power to dispose about the protection of subject rights.

2.3.3 Implications

Commentators and data protection authorities seem in agreement about the intention of Article 25's factors of controller discretion. Controllers should be free to select any measure as long as it is effective in risk mitigation. However, this implies discretionary powers on many levels, ranging from the risk assessment to the determination of state of the art. Most commentators conclude that this margin of discretion causes legal uncertainty and creates a large room for legal dispute. Bieker and Hansen [BH17] see a solution in more technical guidelines provided by data protection authorities. This seems an inevitable and enormous task that will also require academic support to more objectively assess factors like cost and state of the art, which will eventually determine how (much) controllers have to implement data minimisation.

II

Timestamp Usage and Demand

3 Towards Minimising Timestamp Usage In Application Software: A Case Study of the Mattermost Application

Citation	Christian Burkert and Hannes Federrath. *"Towards Minimising Timestamp Usage In Application Software: A Case Study of the Mattermost Application"*. In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Ed. by Cristina Pérez-Solà et al. Vol. 11737. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 138–155. ISBN: 978-3-030-31500-9. DOI: 10.1007/978-3-030-31500-9_9
Ranking	*n.a.* (workshop traditionally co-located with ESORICS)
Status	Published
Publication Type	Research Paper
Aim	The aim of this paper was to explore methods of analysing the status quo of timestamp privacy in application software and to gather first evidence about the potential for timestamp minimisation. The case study should introduce the issue of excessive timestamp collection to the privacy and software engineering community and discuss techniques of applied privacy engineering with respect to a practical data minimisation of timestamp data in application data models.
Methodology	The paper uses a white box analysis of application source code to gather information about the presence of timestamps in the data model as well as their programmatic use. Basic techniques of static program analysis are used to reliably locate uses of given data model attributes in the application code. Different classifications are used to classify timestamp type (keyword based) and usage purpose (manual context analysis with bottom-up classification).

Continued on next page

Contribution	The paper contributes the first systematic analysis of timestamp usage in application software, including a novel yet laborious manual process to gather timestamp type and usage classifications based on source code. It provides suggestions for more data-minimal design alternatives for common functions of timestamps. Furthermore, the provides the first description and empirical evidence for the phenomenon of excessive timestamp inclusion in application data models. To that extend, the case study showed that the investigated app has a high degree of collected but unused timestamp information and that timestamps are processed for purposes which can be fulfilled by more data-minimal alternatives.
Co-authors' contribution	The article was co-authored by Hannes Federrath, who provided editorial guidance.

Abstract

With digitisation, work environments are becoming more digitally integrated. As a result, work steps are digitally recorded and therefore can be analysed more easily. This is especially true for office workers that use centralised collaboration and communication software, such as cloud-based office suites and groupware. To protect employees against curious employers that mine their personal data for potentially discriminating business metrics, software designers should reduce the amount of gathered data to a necessary minimum. Finding more data-minimal designs for software is highly application-specific and requires a detailed understanding of the purposes for which a category of data is used. To the best of our knowledge, we are the first to investigate the usage of timestamps in application software regarding their potential for data minimisation. We conducted a source code analysis of Mattermost, a popular communication software for teams. We identified 47 user-related timestamps. About half of those are collected but never used and only 5 are visible to the user. For those timestamps that *are* used, we propose alternative design patterns that require significantly reduced timestamp resolutions or operate on simple enumerations. We found that more than half of the usage instances can be realised without any timestamps. Our analysis suggests that developers routinely integrate timestamps into data models without prior critical evaluation of their necessity, thereby negatively impacting user privacy. Therefore, we see the need to raise awareness and to promote more privacy-preserving design alternatives such as those presented in this paper.

3.1 Introduction

The ongoing process of digitisation greatly affects the way people work: equipment, work environments, processes and habits. Office workers use centralised software systems to advance collaboration and the exchange of knowledge. Even field workers are in frequent interaction with software systems in their headquarter, e.g., package delivery personnel that reports its current position for tracking and scheduling. All these interactions of employees with software have the potential to create digital traces that document behaviour and habits.

At the same time, there is a demand to use such data about work processes and employees to benefit the company and deploy (human) resources more productively. This interest is commonly referred to as *people analytics* [DiC19; TLP18]. Major software vendors such as Microsoft are already integrating people analytics functionality into their enterprise software products to provide managers and employees with more analytics data [Mic].

On the other hand, employees might object to their personal data being analysed by their employer, e.g., for the fear of being discriminated or stigmatised [Bor18]. In fact, the EU General Data Protection Regulation (GDPR) is also applicable in the context of employment [Ogr17]. It specifies in Article 25 that controllers, i.e., employers, shall implement technical measures, which are designed to implement data-protection principles, such as data minimisation. Thereby, controllers are generally obligated to use software that follows data minimisation principles.

While there has been a lot of work on applying data minimisation to software engineering [Dan+15], no particular focus has been given to minimise a particular kind of metadata: timestamps. We reason that timestamps are a ubiquitous type of metadata in application software that is both very privacy-sensitive and not yet understood regarding its potential for data minimisation.

In combination with location data, it is well understood that timestamps function as so-called quasi identifiers [Swe02] and contribute to the linkability and de-anonymisation of data [Wer+14]. Take for instance the NYC taxi dataset, where prominent passengers could be re-identified from seemingly anonymised taxi logs by correlating pickup location and time with external knowledge such as paparazzi shots [Pan14]. We reason that timestamps should be considered as similarly sensitive outside the context of location-based services as well, especially regarding user profiling through application software. In the latter case, the identifying potential of timestamps is not a prerequisite for them to be a privacy risk, because users are commonly already identified by other means (e.g. credentials). Instead, timestamping actions allows for a temporal dimension in user profiling, which gives deeper insights into behavioural patterns, as one does not only learn where users are going, but also when and in what intervals. For instance, recent work has shown that timestamps in edit logs of real-time collaboration services such as Google Docs are sufficiently detailed for impersonation attacks on typing-biometric authentication systems [MR18].

We use the term personally identifiable timestamp (short: *PII timestamp*) to describe a timestamp that can be directly linked to a person within the data model of an application. Regarding application software, timestamps are a basic data type and their potential applications manifold. However, to design more data-minimal alternatives to timestamps, an understanding about their current usage in application software is required.

To gain insights into possible uses and alternatives for timestamps in application software, we picked Mattermost as the target of evaluation for this case study. Mattermost presented itself as a suitable target because of its open source, centralised data management and popularity as a communication platform for teams and Slack alternative [Mat].

Our main contributions are: (i) We conduct a source code analysis to identify and describe timestamp usage in Mattermost server. (ii) We describe alternative design patterns for each type of usage.

The remainder of the paper is structured as follows: Section 3.2 provides background on our adversary model. Section 3.3 analyses the usage of timestamps in Mattermost. Section 3.4 presents alternative design patterns. Section 3.5 discusses related work and Sect. 3.6 concludes the paper.

3.2 Adversary Model

Our adversary model follows the established honest-but-curious (HBC) notion commonly used to assess communication protocols. Paverd et al. [PMB14] define an HBC adversary as a

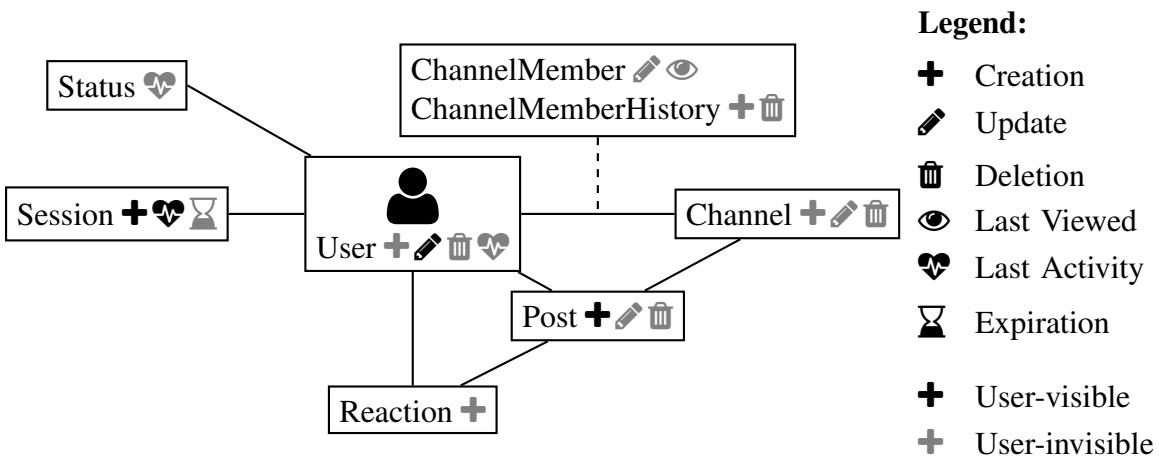


Figure 3.1. Mattermost's core components with their respective timestamps, categorized according to their type and visibility.

legitimate participant in a communication protocol, who will not deviate from the defined protocol but will attempt to learn all possible information from legitimately received messages. Adapted to the context of application software and performance monitoring, we consider an adversary to be an entity that is in full technical and organisational control of at least one component of a software system, e.g., the application server. The adversary will not deviate from default software behaviour and its predefined configuration options, but will attempt to learn all possible information about its users from the collected data. This especially means that an adversary will not modify software to collect more or different data, or employ additional software to do so. However, an adversary can access all data items that are collected and recorded by the software system irrespective of their exposure via GUIs or APIs. We reason that this adversary model fits real world scenarios, because employers lack the technical abilities to modify their software systems or are unwilling to do so to not endanger the stability of their infrastructure.

3.3 Application Analysis

We analysed Mattermost as an exemplary application to gather insights into developers' usage of timestamps. Our analysis uses Mattermost Server version 4.8 released in Nov. 2018 (current version in June 2019: 5.12). The source code has been retrieved from the project's public GitHub repository [Mat18a]. To determine which timestamps are presented to the user, we used Mattermost Web Client in version 5.5.1 [Mat18b].

The analysis is structured as follows: First, we identify timestamps in the source code, then we determine those timestamps that are relatable to users. Subsequently, we investigate their type, user-visibility, and programmatic use, before we discuss our findings.

3.3.1 Identification of Timestamps

Initially, we identify all timestamps that are part of Mattermost's data model code located in the dedicated directory `model`. Therein, we searched for all occurrences of the keyword `int64`, which denotes a 64-bit integer in the Go programming language. This integer type is used by Mattermost to store time values in milliseconds elapsed since January 1st, 1970. From our keyword search, we excluded all test code, which is by Go's design located in files whose filenames end in `_test.go` [Goo19a]. This initial keyword search yielded a list of 126 occurrences which not only contains timestamp-related data model declarations, but also other integer uses and occurrences of the keyword within type signatures.

Criterion	Description	Freq.
Cast	Keyword is used to type cast a variable	12
Signature	Keyword is used within a type signature	7
Local	Keyword is used to declare a local variable	6
Counter	As indicated by the name containing `count`, `sequence` or `progress`	14
Setting	A setting as located in `config.go` or `data_retention_policy.go`	8
Identifier	Used as object identifier as indicated by the name `id`	4
Size	Used to record object sizes as indicated by the name containing `size`	2
Priority	Used as priority level as indicated by the name `priority`	1

Table 3.2. Exclusion criteria for occurrences of the keyword `int64` in Mattermost's data model source code. The top three are syntactical criteria, whereas the remaining are semantic criteria based on indicators in variable and file names.

Based on the list of 126 occurrences of the keyword `int64`, we narrowed down the candidates for timestamps in Mattermost's data model by excluding all occurrences that are syntactically or semantically out of scope. Table 3.2 lists the criteria of exclusion along with the respective frequency of occurrence in our keyword search. In total, 53 occurrences could be excluded due to these criteria. The remaining 73 occurrences showed clear semantic indicators of being timestamp-related, of which the most common indicator was a variable naming scheme in the form a state-defining verb followed by the preposition *at*, e.g., `CreateAt`. This `At`-naming scheme occurred 64 times, followed by the naming-based indicators `time` and `Last...Update` with 7 and 2 occurrences, respectively.

3.3.2 Selection of PII Timestamps

As described before, we limit the scope of our analysis to PII timestamps, i.e., timestamps that mark an event which is directly or indirectly linked to a natural person. Regarding Mattermost, we consider timestamps as personal or PII, if the enclosing composition type also includes

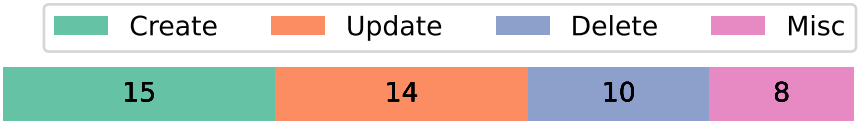


Figure 3.2. Distribution of timestamp types among the PII timestamps.

a direct or indirect reference to the user object, e.g., the creation time of a post is personal because the post object also contains a reference to the creating user.

To determine whether or not the timestamp members identified in Sect. 3.3.1 meet the criteria for PII timestamps, we inspected Mattermost's source code. We conclude that a timestamp is PII, if its composition type, i.e. `struct`, contains the `User` type, or any of the referenced composition types – including their references recursively – contain the `User` type. Out of the 73 timestamps identified in Mattermost's data model, 47 are directly or indirectly linked to a user.

3.3.3 Distribution of Timestamp Types

Having identified the PII timestamps, we analysed the type of these timestamps. By type, we mean the type of event that is recorded in this timestamp, e.g., the creation or deletion of an object. To conduct this analysis, we took advantage of the variable naming scheme mentioned in Sect. 3.3.1, which allowed us to infer the type of the timestamped event from the verb used in the variable name. For instance, we can infer from the variable name `UpdateAt` that this timestamp records the time when the respective object is updated. Figure 3.2 shows the distribution of timestamp types as inferred from their names. The most common types are create and update timestamps, that each make up almost a third of all PII timestamps. Delete timestamps are less frequent and occur only 10 times. We classified timestamps as type *create* if their name contains the word `create` or `join`, as type *update* if their name contains the word `update` or `edit`, and as type *delete* if their name contains the word `delete` or `leave`. The remaining timestamps are classified as *miscellaneous* or *misc*, and include timestamps named `LastActivityAt` (3 occurrences) and `ExpiresAt` (2).

3.3.4 User-visible Timestamps

One possible use of timestamps is to inform users, e.g., about the time when a password has last been changed. To assess how many of the identified PII timestamps serve that purpose, we inspected the graphical user interface of Mattermost's Web Client in version 5.5.1 [Mat18b]. We executed a manual depth-first walk through the graphical user interface starting from the town square channel view. Alternatively, we considered using a (semi-) automatic approach of data flow analysis from the data source (REST API) to the data sink (renderer). However, we abandoned that approach as we found no way to determine all possible sinks.

During the manual GUI inspection, we clicked on and hovered over every apparent GUI element, looking for timestamps that are visible to the user. In doing so, we found the following 5 timestamps that are visible to users without special privileges:

- `Post.CreateAt`: The creation time appears next to the username above a post, or left of the post if the same user posts repeatedly without interruption. The presented time is not altered by editing the post, but remains the creation time. It is visible to all members of the respective channel.
- `Session.CreateAt` and `Session.LastActivityAt`: Both, the time of creation and the time of last activity in a session are shown in the *Active Sessions* that are reachable via the *Security* section in the account settings.
- `User.LastPasswordUpdate`: The time of the last password update is shown in the *Security* section of the account settings dialog. Each user can only see their respective timestamp.
- `User.LastPictureUpdate`: The time of the last picture update is shown in the *General* section of the account settings dialog. Each user can only see their respective timestamp.

Note that the visibility assessment only considers timestamps that are visually rendered as part of the graphical user interface and not timestamps that are readable via an API.

3.3.5 Programmatic Uses of Timestamps

To identify other uses of PII timestamps apart from informing users, we conducted a source code analysis of programmatic timestamp uses. In the following, we first describe the process of source code analysis which we used to locate potential programmatic uses of the identified PII timestamps. Second, we explain the process of classifying uses as programmatic. In short, we consider a use as programmatic, if the value of the timestamp has an impact on the behaviour of the application. Take for instance the usage of a post's creation timestamp that determines if a user is still allowed to edit their post.

Locating Timestamp Uses

The aim of this source code analysis is to find all uses of PII timestamps within Mattermost's server code. To locate all uses of PII timestamps, we used *gorename*, a refactoring tool that is part of the Go tools package [Goo19b]. Gorename is intended as a refactoring tool for type-safe renaming of identifiers in Go source code. We modified gorename to discover and list all occurrences of a given identifier in a type-safe manner, which allows us to automatically determine, e.g., an operation on `o.UpdateAt` as belonging to `Session.UpdateAt` and not `Channel.UpdateAt` solely based on the static typing of `o`. We used this ability to locate all occurrences of PII timestamp identifiers, e.g. `User.CreateAt`, within the server

code base. This yielded a list of file names and line numbers referencing the found location of timestamp uses.

Type of Use	Description
AutoReply	Set date of system-initiated auto-replies
Copy	Copy of object including timestamp
CurAss	Current time is assigned
Definition	Timestamp variable is defined
EditLimit	Enforce edit limit for posts
Etag	Calculate Etag for HTTP header
Expiry	Enforce the expiry of an object
Filter	Filter a sequence of objects by time
Format	Format timestamp for human readability
ImportOld	Support import of old Mattermost data
Inter	Used as intermediary in assignment of another PII timestamp
MinElapse	Ensure that a minimum amount of time has elapsed
EmailDate	Inform about post creation time in an email notification
PostNovelty	Highlight new posts
SetZero	Set timestamp to zero
Sort	Sort a sequence of objects by time
State	Track the state of an object
StateDeleted	Check if an object has been deleted
Timeout	Enforce a timeout
Valid	Validation of timestamp value

Table 3.3. Types of use of PII timestamps within Mattermost's server code. Usage types that we consider as programmatic are highlighted by a grey background.

Classification of Types and Programmatic Uses

Based on the list generated by gorename, we inspected each found use of a PII timestamp identifier and conducted a bottom-up classification, by which we formed groups of uses that fulfil the same or a similar programmatic purpose. The resulting usage type classification is shown in Table 3.3. Following this basic classification, we assessed for each usage type whether or not it constitutes a programmatic use.

We consider a use as *programmatic* if it (a) determines the behaviour of Mattermost, and (b) is not self-referential, i.e., is used for purposes other than maintaining timestamps. For instance, we consider `AutoReply` not as a programmatic use, because the need to derive a date for auto-reply posts only arises from having creation dates in posts in the first place (self-referential). Similarly, the assignment of the current time to (`CurAs`) and the validation (`Valid`) of timestamps are also not programmatic, because both are only necessary to prepare

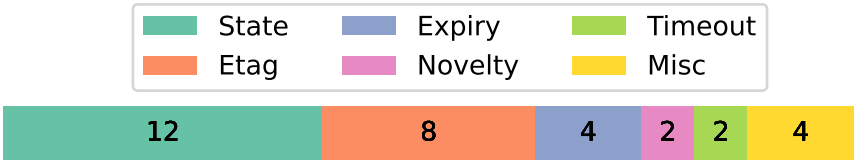


Figure 3.3. Distribution of programmatic uses between the identified usage types.

for other uses. On the other hand, we consider `EditLimit` a programmatic use, because it implements the policy that posts should only be editable for a certain amount of time (determines behaviour). Table 3.3 highlights usage types that are classified as programmatic by a grey background.

Of these 10 types of programmatic uses of PII timestamps, we found 32 instances. Figure 3.3 shows the frequency in which each type occurred. Note that `StateDelete` is included in `State` and is regarded as a special case of the latter from now on. The types that are summarised under miscellaneous are EditLimit, Filter, MinElapse, and Sort, each occurring once.

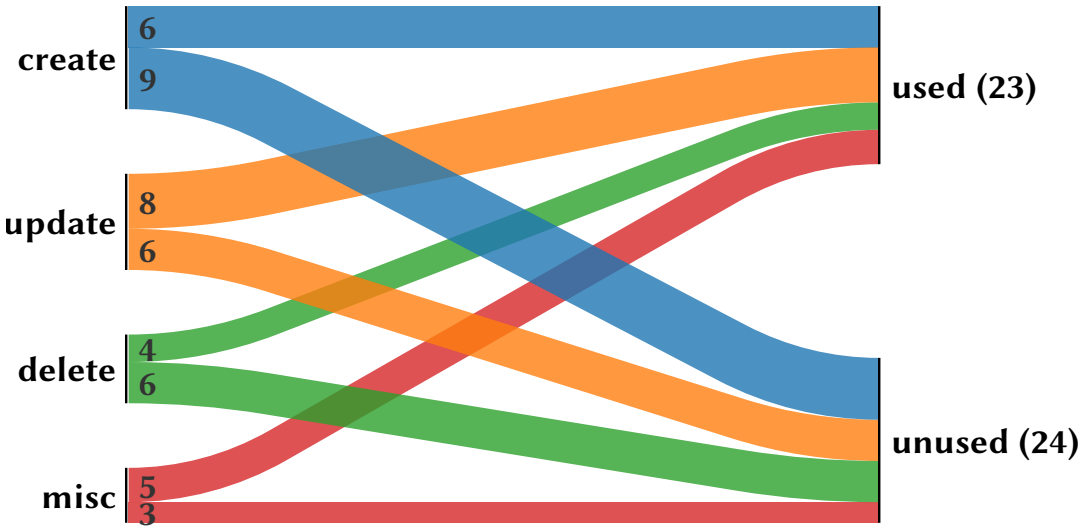


Figure 3.4. Distribution of timestamp types between used and unused timestamps.

Our investigation also showed that 24 out of the 47 PII timestamps have no programmatic use at all. Figure 3.4 shows that the timestamp types are almost evenly distributed between the used and unused timestamps. Only 40% of create and delete timestamps are used, whereas almost 60% of update timestamps are used. The four uses of delete timestamps are all of the type `StateDeleted`, which only checks if the timestamp equals zero or not. Thus, the actual time of deletion is never used programmatically.

3.3.6 Summary

Our analysis indicates that most of the PII timestamps have no purpose because they are neither programmatically used by the application nor presented to the user. This might suggest that developers routinely add these timestamps to a data model without reflecting their necessity. Regarding the programmatic usage of timestamps, we observe that timestamps are used to track intra-object state (ETags, State), for inter-object comparisons (PostNovelty, Sort), to measure the passage of time (EditLimit, Expiry, MinElapse, Timeout), and to allow references with an external notion of time (Filter).

3.4 Privacy Patterns for Timestamp Minimisation

The analysis of timestamp usage within Mattermost identified several types of timestamp usage either programmatic or informative. These types of usage are currently designed to process and present timestamps with a millisecond resolution. In the following, we will present alternative design patterns for these usage types that require less detailed timestamps or none at all.

3.4.1 Notation

When describing the resolution of a timestamp and the reduction thereof, we use the following definitions and notations in the remainder of this paper.

Definition 1 *Given a timestamp $t \in \mathbb{N}$ as seconds since January 1st, 1970 and a resolution $r \in \mathbb{N}$ in seconds, we define the reduction function reduce: $\mathbb{N} \to \mathbb{N}$ as follows: $reduce(t,r) := \left\lfloor \frac{t}{r} \right\rfloor r$.*

Definition 2 *Similarly, given a resolution $r \in \mathbb{N}$ in seconds, we define the set of reduced timestamps $\mathbb{T}_r$, with $\mathbb{T}_r \subseteq \mathbb{N}$, as $\mathbb{T}_r := \{reduce(t,r) \mid t \in \mathbb{N}\}$.*

Note 1 *We use the suffixes h and d to denote an hour or a day when specifying the resolution r. Therefore, $\mathbb{T}_{1\mathsf{h}}$ equals $\mathbb{T}_{3600}$ and $\mathbb{T}_{1\mathsf{d}}$ equals $\mathbb{T}_{24\mathsf{h}}$.*

3.4.2 Expiry and Timeout

Expiry or timeout mechanisms have to decide whether a given amount of time (delta) has elapsed since a reference point in time. This can be required, e.g., to check if a session has reached its maximum lifetime, or to determine whether a user's period of inactivity is long enough to set their status to absent. A naive implementation of this mechanism can either store the reference point and the delta, or the resulting point of expiry. Mattermost, for instance, uses both approaches. A more data-minimal design could reduce the resolution of the reference or expiry point, thereby recording less detailed traces of user behaviour.

3.4.3 Sorting

The purpose of sorting is that a sequence of objects is ordered according to a timestamp. Mattermost uses the creation timestamp of posts to present them in temporal order. Thereby, the distance between the posts' creation timestamp is irrelevant. Instead, only a timestamp's property as an ordering element is used. This functionality can also be realised by using sequence numbers that are automatically assigned to each post upon its creation.

3.4.4 Filtering

Mattermost allows users to filter posts by their update timestamp. A user can request Mattermost to show only posts that were last updated before, after or on a given date. Since filtering is interfacing with users, the user-provided date needs to comply with users' perception of time and allow intuitive date formats. As a consequence, sequence numbers for posts do not directly apply, because users cannot be expected to know or determine the range of sequence numbers that fits their desired date filter.

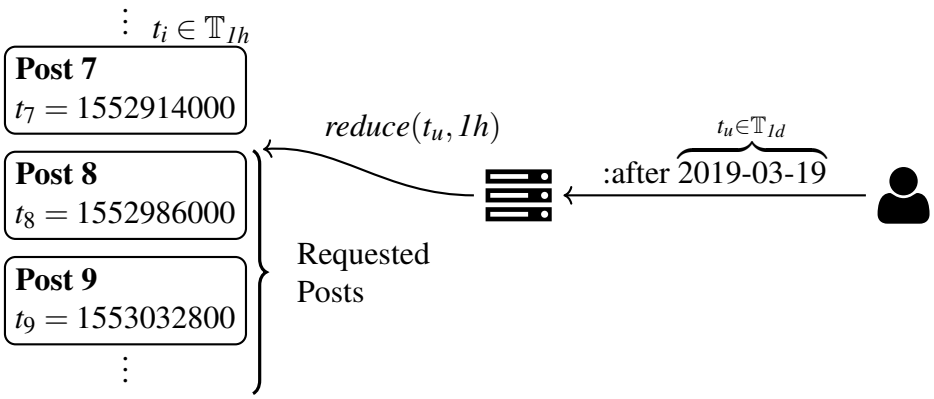


Figure 3.5. Filtering can be run on timestamps with significantly reduced precision. This example illustrates filters with a one-day resolution that are applied on post timestamps with a resolution of one hour.

Instead, we propose using per-post timestamps with a reduced resolution. Figure 3.5 illustrates a simple filter mechanism that stores per-post timestamps with a resolution r_p of one hour. Users can specify a filter date t_u with a resolution r_u of one day. The resolution of t_u is reduced to r_p, if r_p is greater than r_u, which is necessary to ensure the discoverability of all posts within the given filter range. Note that in that case, a reduction of the resolution increases the temporal range of the given filter. As a result, a filter request might return posts that lie outside of the original filter range, thus potentially causing confusion among users, especially if $r_p \gg r_u$. We call this phenomenon *filter range extension*.

Mattermost allows to specify date filters with a resolution of one day ($r_u = 1d$). If per-post timestamps are only used for filtering, then their resolution should be equal or greater than the

filter resolution ($r_p \geq r_u$). Smaller values for r_p, i.e. $r_p < r_u$, would not increase the precision of the filter mechanism, since the overall filter resolution is the minimum of r_p and r_u and thus limited by r_u. Larger values for r_p would enhance privacy protection, but cause for confusing filter range extensions on the other hand, if $r_p \gg r_u$. Therefore, $r_p = r_u$ is a sensible setting.

Note that for sequence numbers to work with filtering, a mechanism would be needed that translates human-understandable timestamp filters into sequence number filters that are comparable to sequence numbers of recorded posts. However, since posts are generally not created in equidistant time intervals, such a translation mechanism would need additional information about the distance between posts as well as at least one absolute point of reference or anchor point, to be able to map a user-given timestamp to a sequence number. Determining the respective sequence number from a given timestamp would require summing up inter-post intervals starting from the closest anchor point. Therefore, we consider sequence numbers as impractical for filtering. Also note that regarding privacy, sequence numbers in combination with anchor points provide no advantage over timestamps with reduced precision.

3.4.5 ETag

An ETag (short for entity-tag) is an HTTP header field and one of two forms of metadata defined in RFC 7232 [FR14] that can be provided to conditionally request a resource over HTTP. It is defined as an opaque string that contains arbitrary data. In contrast to the last-modified header field, an ETag can be created without the use of timestamps. Nevertheless, to fulfil its purpose of testing for updates and validating cache freshness, an ETag should be indicative of state changes and should be generated accordingly. For that purpose, it is convenient to include a last-modified timestamp in an ETag instead of including every state-defining attribute of a resource. However, the same level of convenience can be achieved by using a revision number instead of a last-modified timestamp. Such a revision counter could be added to each data model class that is requestable via HTTP. It would be incremented every time the respective object is changed. RFC 7232 [FR14] itself mentions in section 2.3.1 revision numbers as a way to implement ETags, alongside collision-resistant hashes of representative content and the critiqued modification timestamp.

3.4.6 Novelty Detection

Mattermost uses timestamps to detect and highlight unread posts. Therefore, Mattermost records the last time a user has viewed a channel. Upon revisiting a channel, Mattermost can then simply identify unread posts by comparing their creation timestamp to the channel's last visitation timestamp.

As an alternative, a sequence number could be assigned to each post. Then it would suffice to record, for each channel and user, the sequence number of the most recent post at the time of a user's last visitation. Following this design, unread posts can be identified as those posts whose sequence number exceeds the recorded sequence number of the last read post.

3.4.7 State Management

Mattermost uses timestamps to keep track of objects' state. An unset creation timestamp signifies that an object is not yet fully initialised and an unset deletion timestamp signifies that an object is still active and not yet deleted. However, there is no advantage in using timestamps to record the state of an object, besides saving the amount of storage that would be needed to use a boolean or integer variable in addition to a timestamp. If state management is the only purpose of a timestamp, it can be easily replaced by a boolean or state enumeration variable. In the case of Mattermost, we found that all deletion timestamps and 3 creation timestamps are only used for the purpose of managing state.

3.4.8 User Information

Mattermost presents some timestamps to the user in its graphical user interface (see Fig. 3.1). Only one of them, namely the post creation timestamp is visible to all users, whereas the other timestamps are only visible for the currently logged-in user.

The potential for minimising user-visible timestamps depends on the frequency of the time-stamped event. Take, for example, the creation timestamps of posts: The higher the posting frequency the shorter the interval between posts and the more detail is required for timestamps to be distinctive. Another aspect that influences the potential for minimising user-facing timestamps is the locality of distinctiveness, i.e., the question whether users rather use time-stamp information to distinguish posts in close temporal proximity to each other, or use it to gain a coarser temporal orientation.

Consider a timestamp resolution of 15-minutes: Posts that are created within a 15-minute period would all be presented with the same one or two timestamps, potentially creating a false and confusing impression of immediate succession. Therefore, a user-facing reduction of timestamps needs to be obvious to avoid misinterpretation. This can be achieved by annotating such timestamps accordingly, e.g., by explicitly displaying an interval like *14:30–14:45*.

Besides reduction, user-visible PII timestamps can also be protected by encrypting them in a way that only authorised users can decrypt them. In case of timestamps that only concern a single user, this can be realised with a common asymmetric cryptographic system, where the secret is protected by the user's password. In case of timestamps that should be readable by multiple users, e.g., post creation timestamps, a more complex cryptographic setup is required that also has to handle churn among authorised users.

3.4.9 Compliance

Another reason to record timestamps that is not directly reflected in our analysis might be compliance, e.g., documentation obligations or judicial orders. In such cases, the potential for minimising PII timestamps is limited and depends on specific regulations. However, the

	EditLimit	Etag	Expiry	Filter	MinElapse	Novelty	Sort	State	Timeout	User Info
Sequence Number						●	●			
Revision Number		●								
Reduction	●		●	●	●				●	●
Encryption										●
Enumeration								●		

Table 3.4. Overview of timestamp usage and the respectively suited design alternatives.

impact of recording PII timestamps on user privacy can at least be reduced by restricting access and limiting storage periods. We suggest to store such compliance timestamps separately from production data and encrypt them using a threshold scheme [DF89] in order to separate the duty of decryption among multiple parties for the sake of preventing misuse.

3.4.10 Summary and Discussion

Based on the identified timestamp usage, we were able to present more privacy-preserving alternatives to using full-resolution timestamps. The presented alternative design patterns are constructed of five technical primitives: sequence numbers, revision numbers, reduction of precision, encryption, and enumerations. Table 3.4 gives an overview of the design alternatives and illustrates which of the five primitives are applicable to which timestamp usage.

We find that four of the identified timestamp usage types, namely Etag, Novelty, Sort, and State, can be replaced by sequence numbers, revision numbers, and enumerations. These four usage types together make up more than half of the total number of PII timestamp usage in Mattermost (see Fig. 3.3). For the remaining usage types, sequence numbers are not an option because (a) values need to be comparable to a user-provided date (Filter), (b) values need to be human readable (User Information), or (c) values need to be comparable to an absolute point in time (Expiry, MinElapse, Timeout). In those cases, the privacy impact of recording timestamps can be reduced by reducing their resolution or by encrypting them.

Discussion

Article 5(1)(c) GDPR states that the extent to which the processing of personal data shall be limited is determined by the purposes for which they are processes. Hence, the principle of data minimisation does not demand absolute minimisation but minimisation relative to a given purpose. In that sense, the goal of our alternative designs and minimisation patterns in general is not to eliminate all timestamps from application software, but to replace them with less rich information wherever the latter suffices to fulfil the same purpose.

It should also be noted that the minimisation of personal data can lead to the discrimination of subjects that would not be discriminated otherwise. Take for instance a common data minimisation scheme where postal codes are collected instead of full addresses to determine service coverage. In doing so, a subject might be excluded from a service although they live very close to the serviced area. Hence, such potential negative effects should be taken into account when designing and applying data minimisation patterns.

3.5 Related Work

To the best of our knowledge, we are the first to investigate privacy patterns for the minimisation of timestamps based on their usage in application software. However, there is a rich body of work regarding adjacent topics. In the following, we will present privacy research that focuses on mitigating the privacy impact of timestamps, work regarding the monitoring of employees' performance, and work about the incorporation of data minimisation principles into engineering.

3.5.1 Timestamp Privacy

Basic redaction techniques for timestamps such as reduction and replacement with order-preserving counters have been presented by [SLL06; ZBY06] in the context of log anonymisation. Kerschbaum [Ker07] introduces a technique to pseudonymise timestamps in audit logs as part of a multi-party exchange of threat intelligence data. The introduced technique preserves the distance of the pseudonymised timestamps by using a grid representation. However, the distance calculation requires a third party which is generally not available for our application.

In wireless sensor networks, the term *temporal privacy* describes the effort to prevent a passive network observer from inferring the creation time of an event from the observation time of a related message or signal [Kam+07]. Countermeasures in that area include buffering to break the temporal correlation of creation and observation [Kam+07] and temporal perturbation by adding Laplace noise [Yan+15].

Since timestamps can be easily encoded as integers, building blocks from the area of privacy-preserving record linkage regarding numerical values can be applied to compare timestamps in a privacy-preserving manner [KGV18]. However, this approach also requires a trusted third party other than the data-custodians to achieve its privacy guarantees.

3.5.2 Performance Monitoring and People Analytics

There are several approaches to mine (meta) data to gain insight into work processes and employees. A large body of work uses software developers' commit metadata that is publicly available on GitHub. Eyolfson et al. [ETL11] show that late-night contributions of software developers are statistically buggier than contributions in the morning or during the day. Claes

et al. [Cla+18] investigate working hours of developers to gain insight into work patterns that can foster stress and overload detection. Others use this data to infer developers' personality traits like neuroticism [RN16] or to characterise them more generally [OHM13]. Note that these analyses have been conducted with software contribution metadata. However, they can be applied to transactional metadata in general, including interaction data from Mattermost.

People analytics (PA) promises to optimise business processes and human resource management by gathering and analysing data about how employees work [DiC19]. While PA is a trending topic, empirical evidence for its benefits or even concrete metrics are scarce [TLP18]. Part of PA is the understanding of relationships and collaboration dynamics among employees, including their communication. Interaction graphs can be built based on metadata from collaboration software, which then can be analysed using established graph and network algorithms like community detection [WK19]. Insights from such algorithms might be used to optimise team compositions or to identify candidates for management positions.

An automated and algorithmic assessment of employees also raises legal and moral concerns. Bornstein [Bor18] highlights the conflicts of such algorithmic assessments with anti-discrimination and anti-stereotyping regulation. Regarding data protection regulations, the GDPR also protects the personal data of employees and restricts employers' rights to analyse data [Ogr17], especially regarding automatic decision making [Roi17].

3.5.3 Privacy Engineering

One of the privacy design strategies postulated by Hoepman [Hoe14] is *minimise*. The strategies are meant to guide software architects to achieve privacy by design with their software designs. The *minimise* strategy demands that only as much data is collected and processed as is appropriate and proportional to fulfil a given purpose. Whereas timestamps are certainly appropriately used in Mattermost to fulfil the purposes that we identified in Sect. 3.3, the more privacy-preserving design alternatives presented in Sect. 3.4 indicate that using full-resolution timestamps is not proportional for most purposes.

Privacy design patterns are a way to formulate actionable best practices aiming to achieve privacy by design [LFH17]. These patterns focus on concrete and recurring software engineering decisions. While there are several patterns that detail the aforementioned *minimise* strategy, e.g., the location granularity pattern or the strip-unneeded-metadata pattern [Col+19; Kar+19], there are – to the best of our knowledge – no patterns that focus especially on the minimisation of timestamp usage and the replacement of timestamps by less detailed alternatives.

3.6 Conclusion

In this case study, we analysed Mattermost as an exemplary application, to gain insight into the usage of timestamps. We found that Mattermost's data model includes 47 timestamps that are

directly or indirectly linked to actions of a user. More than half of these timestamps have no programmatic use within the application and only 5 timestamps are visible to the user.

We assume that Mattermost is not a special case, but that the use of PII timestamps is commonly excessive and disproportionate. We further assume that this is no result of ill intent but a result of unconscious programming habits and a lack of awareness for privacy anti-patterns.

To find substitutes for full-resolution timestamps, we investigated the potential for data minimisation relative to the identified purpose of usage. Based on that, we presented alternative design patterns that require less precise or no timestamp information. Following these alternatives, more than half of Mattermost's timestamp usage instances can be replaced by easy-to-implement sequence or revision numbers, whereas the resolution of the remaining timestamps can at least be significantly reduced.

We suggest that future work should investigate software developers for unconscious programming habits of adding unnecessary metadata such as timestamps, and find ways to raise awareness. To further design and provide practical alternatives for user-facing timestamps, a user study about the perception of various timestamp resolutions would provide valuable information for sensible defaults.

Acknowledgements

The work is supported by the German Federal Ministry of Education and Research (BMBF) as part of the project Employee Privacy in Development and Operations (EMPRI-DEVOPS) under grant 16KIS0922K.

References

[Bor18] Stephanie Bornstein. *"Antidiscriminatory Algorithms"*. In: *Alabama Law Review* 70.2 (2018), p. 519.

[Cla+18] Maëlick Claes, Mika V. Mäntylä, Miikka Kuutila and Bram Adams. *"Do Programmers Work at Night or During the Weekend?"* In: ICSE. ACM, 2018. ISBN: 978-1-4503-5638-1. DOI: 10.1145/3180155.3180193.

[Col+19] Michael Colesky, Jaap-Henk Hoepman, Christoph Boesch, Frank Kargl, Henning Kopp, Patrick Mosby, Daniel Le Métayer, Olha Drozd, José M. del Álamo, Yod Samuel Martín, Julio C. Caiza, Mohit Gupta and Nick Doty. *"privacypatterns.org"*. 2019. URL: https://privacypatterns.org.

[Dan+15] George Danezis, Josep Domingo-Ferrer, Marit Hansen, Jaap-Henk Hoepman, Daniel Le Métayer, Rodica Tirtea and Stefan Schiffner. *"Privacy and Data Protection by Design - from policy to engineering"*. In: *CoRR* abs/1501.03726 (2015).

[DF89] Yvo Desmedt and Yair Frankel. *"Threshold Cryptosystems"*. In: *CRYPTO*. Vol. 435. Lecture Notes in Computer Science. Springer, 1989, pp. 307–315.

[DiC19] Michael DiClaudio. *"People analytics and the rise of HR: how data, analytics and emerging technology can transform human resources (HR) into a profit center"*. In: *Strategic HR Review* 18.2 (13th Feb. 2019), pp. 42–46. ISSN: 1475-4398. DOI: 10.1108/SHR-11-2018-0096.

[ETL11] Jon Eyolfson, Lin Tan and Patrick Lam. *"Do Time of Day and Developer Experience Affect Commit Bugginess?"* In: *MSR*. ACM, 2011. ISBN: 978-1-4503-0574-7. DOI: 10.1145/1985441.1985464.

[FR14] Roy T. Fielding and Julian Reschke. *"Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests"*. RFC 7232. June 2014. DOI: 10.17487/RFC7232.

[Goo19a] Google, Inc. *"Go testing package"*. 2019. URL: https://golang.org/pkg/testing/.

[Goo19b] Google, Inc. *"Go Tools gorename command"*. 2019. URL: https://godoc.org/golang.org/x/tools/cmd/gorename.

[Hoe14] Jaap-Henk Hoepman. *"Privacy Design Strategies"*. In: *SEC*. Vol. 428. IFIP Advances in Information and Communication Technology. Springer, 2014, pp. 446–459.

[Kam+07] Pandurang Kamat, Wenyuan Xu, Wade Trappe and Yanyong Zhang. *"Temporal Privacy in Wireless Sensor Networks"*. In: *27th IEEE International Conference on Distributed Computing Systems (ICDCS 2007), June 25-29, 2007, Toronto, Ontario, Canada*. IEEE Computer Society, 2007, p. 23. DOI: 10.1109/ICDCS.2007.146.

[Kar+19] Frank Kargl et al. *"privacypatterns.eu"*. 2019. URL: https://privacypatterns.eu.

[Ker07] Florian Kerschbaum. *"Distance-preserving pseudonymization for timestamps and spatial data"*. In: *Proceedings of the 2007 ACM Workshop on Privacy in the Electronic Society, WPES 2007, Alexandria, VA, USA, October 29, 2007*. ACM, 2007, pp. 68–71.

[KGV18] Dimitrios Karapiperis, Aris Gkoulalas-Divanis and Vassilios S. Verykios. *"FEDERAL: A Framework for Distance-Aware Privacy-Preserving Record Linkage"*. In: *IEEE Transactions on Knowledge and Data Engineering* 30.2 (2018), pp. 292–304.

[LFH17] Jörg Lenhard, Lothar Fritsch and Sebastian Herold. *"A Literature Study on Privacy Patterns Research"*. In: *43rd Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2017, Vienna, Austria, August 30 - Sept. 1, 2017*. IEEE Computer Society, 2017, pp. 194–201. DOI: 10.1109/SEAA.2017.28.

[Mat] Mattermost, Inc. *"Mattermost Website"*. URL: https://www.mattermost.org.

[Mat18a] Mattermost, Inc. *"Mattermost Server v4.8.0"*. 2018. URL: https://github.com/mattermost/mattermost-server/releases/tag/v4.8.0.

[Mat18b] Mattermost, Inc. *"Mattermost Webapp v5.5.1"*. 2018. URL: https://github.com/mattermost/mattermost-webapp/releases/tag/v5.5.1.

[Mic] Microsoft. *"Workplace Analytics"*. URL: https://products.office.com/en-us/business/workplace-analytics.

[MR18] Shane McCulley and Vassil Roussev. *"Latent Typing Biometrics in Online Collaboration Services"*. In: *Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC 2018, San Juan, PR, USA, December 03-07, 2018*. ACM, 2018, pp. 66–76. DOI: 10.1145/3274694.3274754.

[Ogr17] Claudia Ogriseg. *"GDPR and Personal Data Protection in the Employment Context"*. In: *Labour & Law Issues* 3.2 (14th Dec. 2017), pp. 1–24. ISSN: 2421-2695. DOI: 10.6092/issn.2421-2695/7573.

[OHM13] Saya Onoue, Hideaki Hata and Ken-ichi Matsumoto. *"A Study of the Characteristics of Developers' Activities in GitHub"*. In: *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*. Vol. 2. 2013, pp. 7–12. DOI: 10.1109/APSEC.2013.104.

[Pan14] Vijay Pandurangan. *"On Taxis and Rainbows. Lessons from NYC's improperly anonymized taxi logs"*. 21st June 2014. URL: https://tech.vijayp.ca/of-taxis-and-rainbows-f6bc289679a1.

[PMB14] AJ Paverd, Andrew Martin and Ian Brown. *"Modelling and automatically analysing privacy properties for honest-but-curious adversaries"*. Tech. rep. 2014.

[RN16] Ayushi Rastogi and Nachiappan Nagappan. *"On the Personality Traits of GitHub Contributors"*. In: *27th IEEE International Symposium on Software Reliability Engineering, ISSRE 2016, Ottawa, ON, Canada, October 23-27, 2016*. IEEE Computer Society, 2016, pp. 77–86. DOI: 10.1109/ISSRE.2016.43.

[Roi17] Antoni Roig. *"Safeguards for the right not to be subject to a decision based solely on automated processing (Article 22 GDPR)"*. In: *European Journal of Law and Technology* 8.3 (2017).

[SLL06] Adam J. Slagell, Kiran Lakkaraju and Katherine Luo. *"FLAIM: A Multi-level Anonymization Framework for Computer and Network Logs"*. In: *LISA*. USENIX, 2006, pp. 63–77.

[Swe02] Latanya Sweeney. *"k-Anonymity: A Model for Protecting Privacy"*. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.5 (2002), pp. 557–570. DOI: 10.1142/S0218488502001648.

[TLP18] Aizhan Tursunbayeva, Stefano Di Lauro and Claudia Pagliari. *"People analytics - A scoping review of conceptual boundaries and value propositions"*. In: *Int J. Information Management* 43 (2018), pp. 224–247. DOI: 10.1016/j.ijinfomgt.2018.08.002.

[Wer+14] Marius Wernke, Pavel Skvortsov, Frank Dürr and Kurt Rothermel. *"A classification of location privacy attacks and approaches"*. In: *Personal and Ubiquitous Computing* 18.1 (2014), pp. 163–175.

[WK19] Nan Wang and Evangelos Katsamakas. *"A Network Data Science Approach to People Analytics"*. In: *Information Resources Management Journal* 32.2 (2019), pp. 28–51. DOI: 10.4018/IRMJ.2019040102.

[Yan+15] Xinyu Yang, Xuebin Ren, Shusen Yang and Julie McCann. *"A novel temporal perturbation based privacy-preserving scheme for real-time monitoring systems"*. In: *Computer Networks* 88 (2015), pp. 72–88. ISSN: 13891286. DOI: 10.1016/j.comnet.2015.06.007.

[ZBY06] Jianqing Zhang, Nikita Borisov and William Yurcik. *"Outsourcing Security Analysis with Anonymized Logs"*. In: *Second International Conference on Security and Privacy in Communication Networks and the Workshops, SecureComm 2006, Baltimore, MD, USA, August 2, 2006 - September 1, 2006*. IEEE, 2006, pp. 1–9. DOI: 10.1109/SECCOMW.2006.359577.

4 Data Minimisation Potential for Timestamps in Git: An Empirical Analysis of User Configurations

Citation	Christian Burkert, Johanna Ansohn McDougall and Hannes Federrath. *"Data Minimisation Potential for Timestamps in Git: An Empirical Analysis of User Configurations"*. In: *ICT Systems Security and Privacy Protection*. Ed. by Weizhi Meng, Simone Fischer-Hübner and Christian D. Jensen. IFIP Advances in Information and Communication Technology. Cham: Springer International Publishing, 2022, pp. 323–339. ISBN: 978-3-031-06975-8. DOI: 10.1007/978-3-031-06975-8_19
Ranking	CORE 2021 B ERA 2010 B QUALIS 2016 B1
Status	Published
Publication Type	Research Paper
Aim	The aim of this paper was to provide the first empirical evidence about users' demand for the precision of timestamps in collaborative software. The understanding of users' demand and expectation for timestamp precision is necessary to determine a proportionate level of recorded timestamp precision that balances actually needed utility with mandated data minimisation. This paper therefore uses publicly documented user preferences for the tool Git to derive their underlying precision demand. Moreover, it aimed to provide empirical evidence for actual privacy gains that can be achieved by applying various levels of timestamp precision reduction.

Continued on next page

Methodology	The paper conducts an empirical analysis of user configuration files published on GitHub. Therefore, the relevant features are automatically extracted, which in combination with verified knowledge of the behaviour of Git, can be used to deduce the minimum level of timestamp precision necessary to fulfil a given user preference. Additionally, the evidence about achievable privacy gains is gathered by an empirical analysis of a large-scale snapshot of public Git commit activity on GitHub. Through data analysis, the privacy gain is evaluated depending on the applied precision reduction.
Contribution	This paper contributes the first empirical evidence about users' demand for timestamp precision in Git. The analysis suggest that a large majority of user preferences can be fulfilled without the need to keep second-precise timestamp information. Moreover, it demonstrates that comparatively small reductions in timestamp precision can already result in a significant privacy gain.
Co-authors' contribution	Johanna Ansohn MacDougall assisted during the writing of the paper and provided editorial support. Hannes Federrath provided editorial guidance.

Abstract

With the increasing digitisation, more and more of our activities leave digital traces. This is especially true for our work life. Data protection regulations demand the consideration of employees' right to privacy and that the recorded data is necessary and proportionate for the intended purpose. Prior work indicates that standard software commonly used in workplace environments records user activities in excessive detail. A major part of this are timestamps, whose temporal contextualisation facilitates monitoring. Applying data minimisation on timestamps is however dependent on an understanding of their necessity. We provide large-scale real-world evidence of user demand for timestamp precision. We analysed over 20 000 Git configuration files published on GitHub with regard to date-related customisation in output and filtering, and found that a large proportion of users choose customisations with lower or adaptive precision: almost 90% of chosen output formats for subcommand aliases use reduced or adaptive precision and about 75% of date filters use day precision or less. We believe that this is evidence for the viability of timestamp minimisation. We evaluate possible privacy gains and functionality losses and present a tool to reduce Git dates.

4.1 Introduction

In increasingly digital work environments, employees' digital and non-digital work steps leave traces of their activities on computer systems. Employers, supervisors and analysts see such data as a resource and opportunity to gain intelligence for business optimisation. Without strong consideration of employees' right to privacy, such legitimate interests might easily lead to excessive and invasive monitoring, even without the employees noticing. Recent reports about mass lay-offs at the game design company Xsolla show that automatic monitoring of employee performance based on software activity logs is already done in practice [Gam21]. Such invasions of employee privacy are however restricted by data protection regulations like GDPR, which requires that the processing of personal data is necessary and proportionate for and limited to the intended purpose.

Software design can contribute to the protection of employee privacy by reducing the amount and detail of data that is stored about user interaction to such a necessary minimum. Prior work, however, indicates that software commonly used in workplaces records especially timestamps in excessive detail [BF19]. It shows that timestamps are not only often unused, but might otherwise also be of unnecessarily high precision. As timestamps allow an easy temporal profiling of employee activities, a reduction in precision could directly reduce the risk of profiling-related discrimination. For instance, a reduction can prevent the inference of intervals between successive work steps and thus mitigate the monitoring of speed and performance. Identifying the necessary level of precision is, of course, dependent on the domain and respective user demand. Nonetheless, similar precision demands can be expected for interactions of similar kind and frequency. In that sense, insights into which levels of timestamp precision are selected by workers if they have the choice, can inform the selection of more appropriate default

precisions in software design. We argue that when users configure their tools to precisions that are lower than the default, this implies that the lower precision is still sufficient for them to fulfil their tasks. Therefore, user customisation is an indicator for users' demand for timestamp precision. With an informed understanding of users' demands, developers can then build software with demand-proportionate timestamping and privacy-friendly defaults.

To the best of our knowledge, we provide the first large-scale real-world analysis of user demand for timestamp precision. Our analysis is based on configuration files for the popular revision control system Git, that users have made publicly available on GitHub. Git is a standard tool for software development workers and its recording of worker activity in the form of commits, contributes significantly to the overall traces that developers leave during a workday. Commit dates have been used to infer privacy sensitive information like temporal work patterns [Cla+18] and coding times [WZ19]. The analysed configurations can contain various preferences that customise the way Git presents dates, including their precision. For instance, using the date formats *iso* or *short* would indicate a high (second) or low (day) precision demand respectively. We also examined the precision of filters (e.g., *8 hours ago*) used to limit the range of outputs. In total, we analysed over 20 000 configuration files. We make the following contributions:

- We compile and provide a comprehensive large-scale dataset of date-related usage features extracted from publicly available Git configs.
- We provide empirical evidence for the demand of date precision by users, as determined by the precision of user selected date formats.
- We discuss and evaluate privacy gain and functionality loss.
- We present a utility that allows users to redact their Git timestamps.

The remainder is structured as follows: Section 4.2 presents related work. Section 4.3 provides a necessary background on Git and its date handling. We describe the acquisition and analysis of our Git config dataset in Sect. 4.4 and Sect. 4.5, and discuss findings, issues and applications in Sect. 4.6. Section 4.7 concludes the paper.

4.2 Related Work

To the best of our knowledge, we are the first to gather empirical evidence for the potential of data minimisation in timestamps. In prior work, we inspected application source code in order to assess the programmatic use of timestamps in application data models [BF19]. The case study of the Mattermost application found that most user-related timestamps have *no* programmatic use and only a small fraction are displayed on the user interface. We addressed the potential to apply precision reduction to user-facing timestamps. However, the code analysis could not provide any indication of acceptable levels of reduction. More work has been done on the exploitation of Git timestamps and the potential privacy risks. Claes et al. [Cla+18] use commit dates to analyse temporal characteristics of contributors to software projects. Eyolfson et al. [ETL11] use dates to find temporal factors for low-quality contributions. Wright and Ziegler

Listing 4.1. Examplary Git config

```
[alias]
  ly = log --date=human --since=yesterday
[blame]
  date = short
[pretty]
  my = %h %an (%ai)
```

[WZ19] train probabilistic models on individual developers' committing habits in an effort to remove noise from coding time estimations. Traullé and Dalle [TD18] analyse the evolution of developers' work rhythms based on commit dates. Following a more general approach, Mavriki and Karyda [MK18b] analyse privacy threats arising from the evaluation of big data and their impact on individuals, groups and society. Drakonakis et al. [Dra+19] evaluate privacy risks of meta data with a focus on online activity in social media and, e. g., try to infer location information from publicly available data. No work seems to exist that proposes or evaluates temporal performance metrics. Slagell et al. [SLL06] proposes time unit annihilation, i. e., precision reduction, to make timestamps less distinct and sensitive. Looking at developer behaviour, Senarath and Arachchilage [SA18] found that while developers typically do not program in a way that fulfils data minimisation, being made aware of its necessity made them apply the principle across the whole data processing chain. With this paper, we also strive to raise the awareness for minimisation of temporal data.

4.3 Theoretical Background: Git and Date Handling

This section provides a background on Git's time and date configuration options. Experts in Git and its date and pretty formatting may jump directly to Sect. 4.4.

Git's command line interface exposes individual actions like creating a commit or listing the history via subcommands like `git commit` or `git log`. Their behaviour can be configured via command line arguments and—to some extent—via settings made in configuration files. Frequently used combinations of subcommands and arguments can be set as shortcuts via so-called *aliases*, similar to shell aliases. For example, the shortcut `git ly` set in Listing 4.1 configures the `log` subcommand to list all commits since yesterday.

In the following, we describe the role and creation of dates in Git and then explain the date-related features that will be empirically analysed later.

4.3.1 Dates in Git

Git associates two types of dates with each commit: the author date and the committer date. Both are usually automatically set to the current date and time, except in case of operations that

Name	Suffix	Precision	Example(s)
default	-	second	Wed Sep 22 14:57:31 2021 +0200
human	-	day to second	Sep 21 2021 / 7 seconds ago
iso	i/I	second	2021-09-22 14:57:31 +0200
raw	-	second	1632315451 +0200
relative	r	year to second	7 years ago / 7 seconds ago
rfc	D	second	Wed, 22 Sep 2021 14:57:31 +0200
short	s	day	2021-10-04
unix	t	second	1632315451

Table 4.2. Git's built-in date formats and their precision.

modify existing commits (e.g., rebases or cherry picks): Here, only the committer date will be updated, but the author date stays as is. Consequently, the author date reflects the time of an initial composition, while the committer date reflects the time of an insertion in the history. Both dates are recorded as seconds since the Unix epoch. Changes to the date precision are not supported by Git. For commit creation, users can provide custom dates through environment variables to use instead of the current. This interface could be used by users to manually set dates with reduced precision. This is however not supported for commands that modify commits in bulk. Here, precision reduction is only possible after the fact, by rewriting the history.

4.3.2 Features for Date Presentation and Filtering

Date Formatting

Date formatting is available for subcommands like `log` and `show` for commit history information, and also for commands that annotate the content of tracked files with commit metadata like `blame` or `annotate`. The formatting option customises how Git renders author and committer dates in the command outputs.

Git offers built-in date formats listed in Table 4.2 and the option for custom format strings which are passed to the system's strftime implementation. The chosen format influences the precision of the displayed date. Five of the eight built-in formats show the full second precision but in different styles like ISO 8601. The others reduce the displayed date precision: *short* omits the time, and both *human* and *relative* use variable precisions that are exact (to the second) when the respective date is recent, and gradually less precise with growing temporal distance. Date formats can be set via the command line option −−date or via config settings for the `log` and `blame` family of subcommands.

Built-in	full	oneline	short	medium	reference	email	mboxrd	fuller	raw
Dates used	none			author				both	
Date Format	-	-	-	default	short	rfc	rfc	default	raw
Fixed Format	-	-	-	-	-	✓	✓	-	✓

Table 4.3. Git offers predefined (built-in) pretty formats that vary in which dates they show and with what date format (Table 4.2) those are formatted by default. Some built-ins are fixed to that default and can not be changed by date options.

Pretty Formatting

Pretty formatting allows the customisation of commit metadata presentation by commands like `log` or `show`, including the names and email addresses of author and committer as well as the dates mentioned above. Like for date formatting, Git offers built-in formats as well as custom format strings with placeholders for each available piece of commit metadata.

Each built-in implies which dates are used (author, committer, or both) and a date format, that—with some exceptions—can be adapted via date options (see Table 4.3). In custom formats, the relevant placeholders are %ad for author dates and %cd for committer dates. The built-in date formats (cf. Sect. 4.3.2) are available as modifiers. For instance, %cr will set the committer date in the *relative* format. Hence, placeholders offer a way to adjust dates separately for each type and independently of other configurations. As shown in Listing 4.1, custom pretty formats can also be set as aliases in a config.

Date Filtering

Some Git subcommands offer limiting their output based on temporal constraints. By passing `--since` or `--until` to log, it will list only commits committed since or until the given reference. References can be given in a wide range of syntaxes and formats, as absolute points in time, time distances, and combinations thereof (e. g., *April 2020*, *01/01 last year*, or *8 hours ago*). Git understands a set of common temporal reference points like *midnight* or *yesterday*. We call those *points of reference*.

4.4 Dataset Acquisition

The basis for our analysis of timestamp precision demand are Git configuration files (configs). To the best of our knowledge, there was no previously available dataset of Git configs or derivations thereof. For that reason, we compiled a dataset based on configs that users published on GitHub. This section describes the identification of the relevant files, their extraction, de-duplication, and the subsequent feature extraction. We also discuss ethical and privacy concerns.

4.4.1 File Identification and Extraction

Git supports a multi-level hierarchy of configs from the individual repository level to the user and system level [Git22, git-config]. We limited our data acquisition to user-level configs, in which users typically set their personal preferences and customisations. These configs are located as `.gitconfig` in the user's home directory. GitHub recognises these files in repositories hosted on their service and assigns them the *Git Config* content language tag. To perform automatic searches, we used the code search endpoint of GitHub's REST API [Git21c]. Due to strict rate limiting and the high load that large-scale code searches might induce on GitHub's servers, we added another condition to narrow down the search to configs that include alias definitions. The resulting search is `alias language:"Git Config"`. We ran the acquisition on Sep 17th, 2021. It yielded 23 691 matching files.

The code search API returns a paginated list including URLs to access the matching files. To obtain all files, we had to overcome GitHub's limitation to return at most 10 pages per query with 100 items each. To do so, we built a crawler that finds small enough sub-queries by using file size constraints. We found that result pages are not necessarily filled to their full 100 items, probably due to pre-emptive processing. We extracted 20 757 matches (88%).

4.4.2 De-duplication

Users might include the same config multiple times in the same or different repositories. To avoid an overrepresentation of users through duplicates, we compared the cryptographic hashes of each config: If a hash occurred more than once within the namespace of the same user, we included only one instance in our dataset. If namespaces differed, we included both instances. We argue that the latter does not constitute a duplicated representation, but a legitimate appropriation by another user and should therefore be counted. Also note that GitHub's Search API does not index forks unless their star count is higher than their parent's [Git21c]. Forks do therefore not introduce unwanted duplicates to our dataset.

We identified and excluded 345 duplicates, i. e., configs that occurred repeatedly in the same namespace. We noticed 554 re-uses of configs in different namespaces. After de-duplication, our dataset comprised configs from 19 468 unique users. 695 users (3.6%) contributed multiple non-identical configs, which accumulate to 1637 configs (8.4%). We argue that those configs should not be excluded, as users might use different configs in different contexts to serve different use cases. Hence, we include them to capture as many use cases as possible.

4.4.3 Feature Extraction

Having extracted the content of matching configs from GitHub, we subsequently tried to parse each config and extract usage information about the date-related features described in Sect. 4.3. In the following, we first describe our process of finding and verifying Git subcommands that support the features in question. After that, we briefly describe the extraction result.

	annotate	blame	diff-tree	log	rev-list	shortlog	show	whatchanged
date	✓†	✓	✓*	✓	✓		✓*	✓
pretty			✓	✓	✓		✓	✓
since/until		✓		✓	✓	✓	✓*	✓

*=undocumented, †=disfunctional

Table 4.4. Support for date-related features in Git subcommands.

To ensure that all relevant subcommands were regarded during extraction, we first searched and inspected the manual pages of each subcommand for the respective command line options. To compensate for any incompleteness or incorrectness in the manuals, we performed automatic tests to check whether any of the date-related options are accepted *and* make a difference to the resulting output. As a result, we identified discrepancies in terms of undocumented feature support as well as non-functioning documented support (in Git version 2.29.2), which we extracted nonetheless. The feature support is shown in Table 4.4.

We performed the feature extraction on all downloaded configs. 41 configs could not be parsed due to invalid syntax. The extraction result comprises usage counts for all options described above as well as derived precision information. We have made the dataset available on GitHub [Bur22].

4.4.4 Potential Ethical and Privacy Concerns

Compiling a dataset of users' Git configurations might raise concerns about the ethics of data extraction or user privacy. We carefully designed our process to address potential concerns.

Code search queries cause a higher load for GitHub than other requests. However, using it enabled us to significantly reduce the total number of queries compared to alternative approaches like searching for repositories named `dotfiles` (about 150 000 matches). Conducting a pure filename-based search is also only possible via code search and would have yielded more than three times as many results without the constraint of the `alias` keyword. We are therefore convinced that our approach minimised the load on GitHub compared to other approaches. In general, we followed GitHub's best practices [Git21b].

Regarding user privacy, our dataset only includes configs that users made public on GitHub. The common practice of publishing *dotfile* repositories follows the spirit of sharing knowledge with the community and providing others with a resource of proven configurations. However, we cannot rule out that an unknown proportion of users uploaded their configuration accidentally or not knowing that it will be public. Our extraction is therefore designed to only extract feature usage counts and no free-form data. This ensures that *no* identifying information, sensitive data or unwanted disclosures like cryptographic secrets are included. We thus deemed it unnecessary to seek approval from the university ethics board.

Aliases	Mean	Std.	Q_1	Q_2	Q_3
subcommand	15.5	22.4	4	9	19
- shell	3.5	8.6	0	1	4
- log-like	2.3	3.1	1	1	3
- blame-like	0.0	0.2	0	0	0
- filter capable	2.3	3.3	1	1	3
pretty format	0.0	0.3	0	0	0

(a) Descriptive statistics about the absolute frequencies of subcommand aliases with sub-types as well as pretty format aliases per config. (Q_i: i-th quartile)

in %	total	date	pretty	since
annotate	28	0.0	-	-
blame	396	2.0	-	0.0
diff-tree	169	0.0	0.0	-
log	41 467	23.7	76.4	2.5
rev-list	194	0.0	5.7	0.5
shortlog	2202	-	-	4.8
show	3440	9.0	19.0	0.1
whatchanged	553	3.6	14.7	-

(b) Relative frequency of feature usage in aliases for subcommands (if supported).

Table 4.5. Alias definitions and date-related feature usage in the dataset.

4.5 Data Analysis

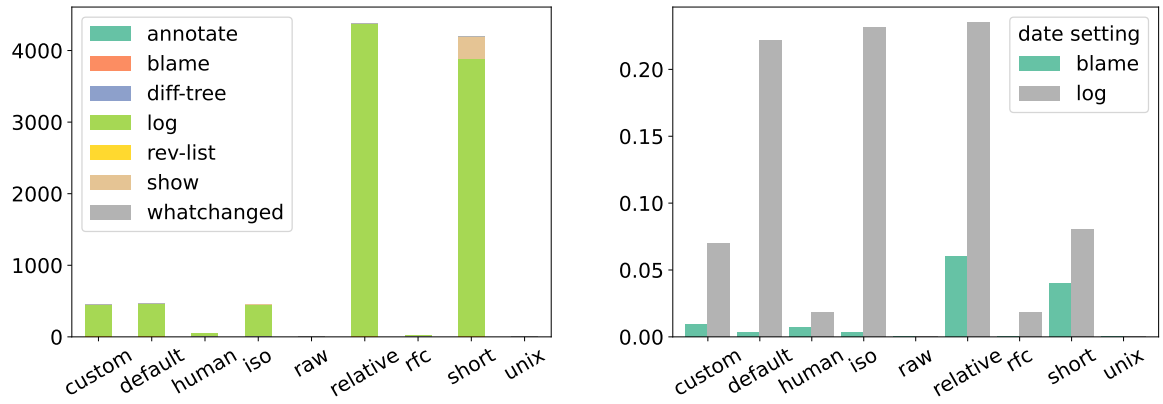
The extracted config features are the basis for our analysis described in this section. To put the findings into perspective, we first provide some basic statistics about the composition of our dataset, before we then analyse users' choices for formatting and filtering, and derive date precisions from them.

We extracted features from 20 369 files. Table 4.5a provides descriptive statistics about the per-config frequencies of subcommand and pretty format aliases. Overall, we extracted 315 520 definitions of subcommand aliases. On average, each config provided 15.5 such aliases. We identified and excluded in total 71 727 (23%) aliases with shell expressions. Table 4.5b provides relative occurrences of features accumulated per subcommand. We found that pretty formatting is very commonly used for log (76%), but less for other subcommands. Date formatting is used fairly often for log as well (24%), but far less for others.

4.5.1 Date Formatting

We analysed the usage of date formats in subcommand aliases and the two config settings for `log` and `blame`. The usage in subcommand aliases is dominated by aliases for `log` (see Fig. 4.1a), since almost a quarter of the more than 41 000 `log` aliases use date formatting. The prevalent formats are *relative* and *short*. Aliases for the `show` subcommand predominantly use the *short* option.

We saw a low use of the settings log.date and blame.date. Only 491 log and 68 blame date formats were set by users, less than 10% of which are custom format strings (see Fig. 4.1b). The *relative* format is again the most popular, but in contrast to command aliases, *short* is only forth after *iso* and *default*. Note that the high number for the *default* format is due to users



(a) used with −−date in command aliases

(b) used for config settings

Figure 4.1. Distribution of date formatting options used as arguments in subcommand aliases or as config settings. The most frequent options vary noticeably between those contexts: *short* is much less common in settings than in aliases.

		built-in [%]				custom [%]			
	total	none	author	committer	both	none	author	committer	both
command aliases	32430	31.1	0.5	0.0	1.3	9.9	28.1	27.3	1.5
pretty aliases	594	0.0	0.0	0.0	0.3	15.2	48.0	30.0	6.6
format.pretty	603	10.0	2.0	0.0	6.6	1.3	48.6	29.4	2.2

Table 4.6. Date usage in pretty formats across all configuration options.

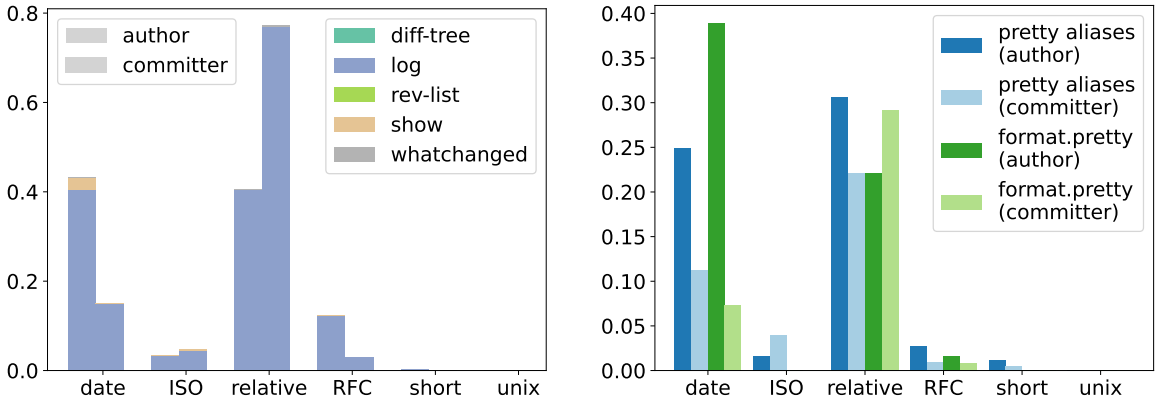
having selected the *default-local* option. As localisation does not factor into date precision, we have counted all localised options for their non-localised correspondent.

Due to the overall low adoption of custom date format strings in conjunction with their comparatively complex and system-dependent interpretation, we omitted them from further analysis.

4.5.2 Pretty Formatting

Date Usage

We analysed which (if any) date types are used in pretty formats for command aliases, pretty .∗ aliases, and the format.pretty setting. For built-in formats, we classified the date use according to documentation [Git22, git-log], which we verified experimentally. For custom formats, we considered a date type as present if the corresponding placeholder (e.g., %ad for author) is contained and not escaped. If we encountered the use of a user-defined format alias, we resolved



(a) custom formats in subcommand aliases

(b) format aliases and `format.pretty`

Figure 4.2. Relative distribution of date modifier usage in pretty formatting. The *relative* and *date* modifiers are most commonly used across all config options.

the alias and proceeded as if the resolved format was directly used. The results are shown in Table 4.6 and described in the following.

Pretty usage in **command aliases** is also dominated by `log`. Over three quarters of all `log` aliases use pretty formatting. The largest proportion of these formats use no dates at all. 31% are built-ins with no date and about 10% are date-less custom format strings. We found that `oneline` with over 90% is the only built-in with frequent use in aliases. Custom formats with either author or committer date are used in around a quarter of formats each. Formats with both dates make up less than 3% combined. Regarding **pretty format aliases**, we extracted 594 uses, which are almost entirely user-defined formats. Their date usage varies significantly from subcommand aliases: Almost half the formats exclusively use author dates and 30% exclusively use committer dates. 15% contain no dates. Regarding **format.pretty settings**, we found 603 configs that use this feature. Here, most occurrences of built-in formats have no (10%) or both dates (7%). The usage of exclusive author and committer dates in custom formats closely corresponds to our observation for pretty aliases, with about 80% combined. However, only about 1% are custom formats with no date. It appears that demand for date-less formatting is satisfied by built-in formats.

Date Modifiers

As described in Sect. 4.3.2, custom pretty formats in addition to choosing the desired date type, also allow a rudimentary date formatting.

For **command aliases**, *date* and *relative* are the most common modifiers (see Fig. 4.2a). More than 75% of committer and 40% of author dates use the *relative* modifier. The *short* date format receives almost no usage. The *date* modifier, which makes the output dependent on `--date` and related settings, is used for about 40% of author dates and about 15% of committer dates. The

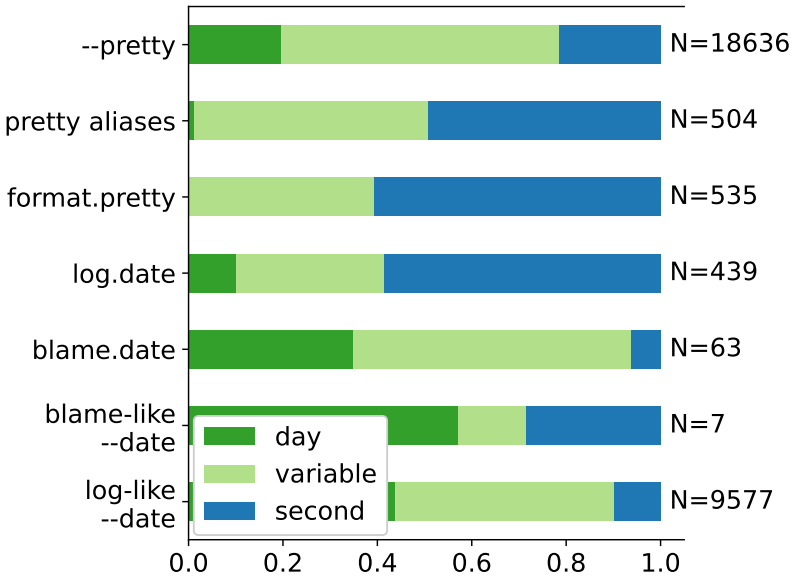


Figure 4.3. Distributions of date output precision across the formatting options. Second precision is least common in alias definitions for log-like subcommands (regarding `--pretty` and `--date`) which also have the most frequent use.

modifier usage in **pretty format aliases** is illustrated in Fig. 4.2b (blue bars). Most used is the *relative* format with combined over 50%, followed by the adaptive *date* modifier with about 35%. The usage in **format.pretty settings** is depicted by the green bars in Fig. 4.2b. Similar to format aliases, *date* and *relative* are by far the most used modifiers with about 40 and 55% each.

4.5.3 Resulting Date Output Precisions

Based on the previous analysis, we could determine the precisions of dates displayed as a result of using the extracted configs. The precision directly follows from the used date format (see Table 4.2) and can be either *second*, *day*, or *variable*. The effective precision of variable formats depends on the recency of the event and ranges between day and second. As we cannot resolve this variability, we will leave it as a third precision in between. Formats with *no* dates are not considered in this section.

For **subcommand aliases** supplying explicit `--pretty` formats, we proceeded differently for built-in and custom formats: For built-ins a with fixed date format, the precision directly follows from the fixed format. For instance, `email` hard-codes the *rfc* format which has a precision of seconds. The precision of all other built-ins is determined by the `--date` option, or—if none is given—by the `log.date` setting or its default, the *default* date format. The same date format resolution applies, if custom pretty formats use the *date* modifier. Otherwise, the resulting precision directly follows from the used modifier. Note that the about 1.5% of custom pretty

formats that use both date types could therefore use a different precision per type. In that case, we considered the higher precision of the two for our further analysis. For that purpose, we used the following sort sequence of precisions: day $<$ variable $<$ second. Figure 4.3 illustrates, e. g., that 60% of subcommands' pretty formatting that contains date information effectively display it with a variable precision. And over 90% of pretty-capable (log-like) aliases that supply −−date options display with variable or day precision.

We applied the same evaluation to pretty formats set as **format aliases** and the **format.pretty** option. Since both settings are taken outside the context of a command invocation, considering possible −−date options is not applicable. Otherwise, we followed the process described above to determine the applicable date format, including considering potential `log.date` options. In contrast to subcommand aliases, day precision outputs are negligible and second precision output is much more common with about 50 to 60% (see Fig. 4.3).

4.5.4 Date Filters

We found that date filter usage is again dominated by `log`. In general, it appears to be an infrequently used feature, with only 2.5% among `log` aliases. In relative terms, it is most commonly used in `shortlog` aliases. We also found that among the date filtering options, −−since makes up for almost the entire feature usage, whereas −−until is almost exclusively used in combination with since. All figures include the alternative names −−after and −−before.

Extraction Methodology

In contrast to date formatting, the precision of filters does not follow directly from a set of predefined options. Moreover, the leniency of the filter parser makes it difficult to cover all allowed inputs during precision classification. For that reason, we decided to directly use Git's parser code for our analysis. We sliced the responsible functionality from the official Git source code [Git21a, v2.32.0-rc2] and linked the functions with our analysis tool. We instrumented Git's date parser at 25 locations to keep track of the smallest unit of time addressed by a filter. This is illustrated by the following two examples:

$$\underbrace{\text{1 hour}}_{\text{hour}}\ \underbrace{\text{30 minutes}}_{\text{minute}}\ \text{ago} \qquad \underbrace{\text{yesterday}}_{\text{day}}\ \underbrace{\text{5pm}}_{\text{hour}}$$

In the first example, the smallest unit is given in minutes, so we consider the filter to have minute precision. In the second example, the smallest unit is given by the full hour, thus we consider the filter to have hour precision. We excluded date filters containing shell command substitutions, of which we identified 26 (2.2%). Another 7 were rejected by Git as invalid, leaving 1156 valid filters.

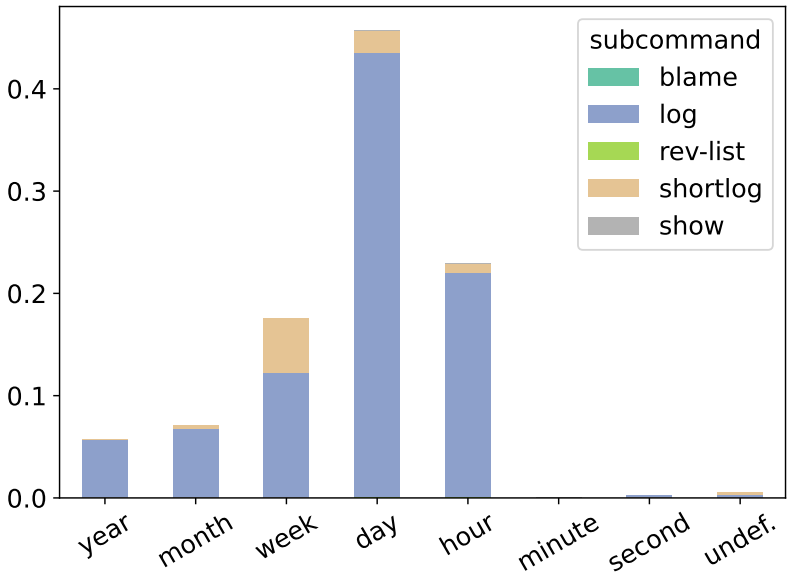


Figure 4.4. Overall distribution of precisions that are implied by the date filters used for the −−since or −−until options in subcommand aliases. Day precision is by far the most common. Filters with precisions higher than hour have almost no use at all.

Precision Classification

When classifying date filter precision, the question arises whether the hour 0 should be treated as hour precision like every other hour value, or as an indicator for the lower day precision. In order to not underestimate the demand for precision, we assumed the hour precision. This is also in concordance with the *midnight* point of reference (POR). The available precision levels are the date unit based precisions year to second (including week), supplemented by the *undefined* precision which is assigned if date filters use the PORs *now* or *never* which allow no classification. Figure 4.4 illustrates the resulting precisions. Most date filters are in the *day* precision (46%), followed by *hour* (23%) and *week* (18%). Precisions higher than *hour* make up less than 0.5%.

4.6 Discussion

In the following, we discuss the privacy gain and functionality loss related to precision reduction, as well as possible objections to the representativeness of our dataset. Additionally, we present options to reduce date precision in Git.

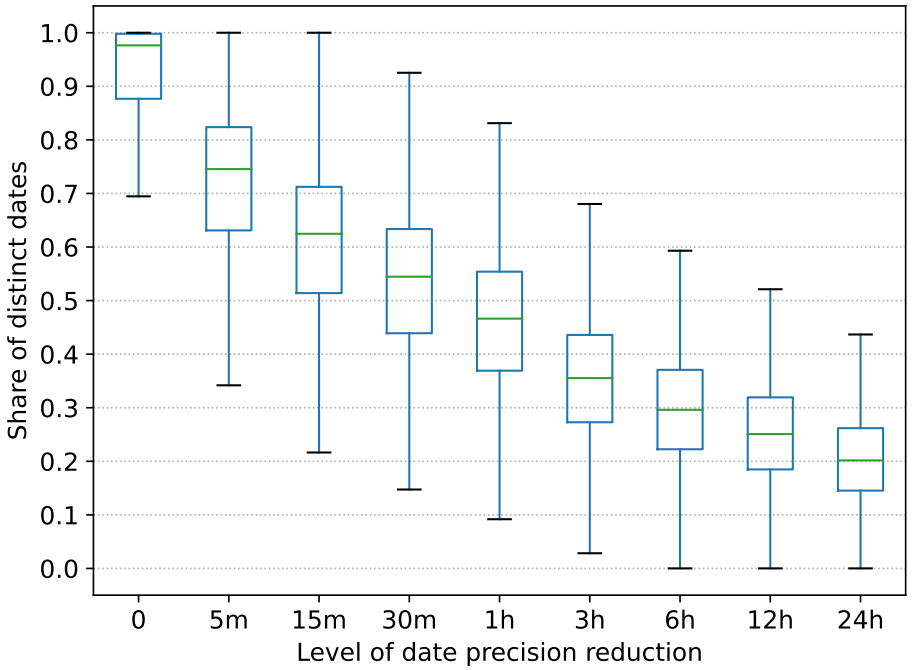


Figure 4.5. The share of users' distinct Git dates decreases fast with increased precision reduction, as evaluated on a GitHub snapshot of 360 million commits. At a 1 hour precision level, more than half of the median user's timestamps are indistinguishable from their chronological predecessors, thus preventing the inference of the temporal intervals between the respective activities.

4.6.1 Privacy Gain and Functionality Loss

In principle, the GDPR mandates data minimisation regardless of achievable privacy gains. Legally, the necessity of data needs to be argued and not its harmfulness to privacy. Nonetheless, to evaluate technical minimisation approaches, some notion of privacy gain might be of interest. Benchmarking the effectiveness of timestamp precision reduction based on known inference techniques is however highly context specific and ignorant to future technical developments. Moreover, timestamp-specific inference techniques are scarce (cf. Sect. 4.2). Instead, we argue that a data-oriented evaluation of statistical properties like changes in distribution are more conclusive of discriminatory power and minimisation effects. For instance, the number of distinguishable activity points over a given period expresses an attacker's ability to observe intervals between actions, which might be used to monitor users' throughput. Following that method, we evaluated the effect of different precision reductions on real-world Git data. This provides additional empirical evidence on the question whether a reduction within the precision range of our previous demand analysis, i. e., no less than day precision, could meaningfully improve user privacy.

We obtained commits from a GitHub mirror curated by the GHTorrent project [Gou13], which contains all public GitHub activity since 2012. Based on a snapshot from April 1st, 2019, we extracted all commits from users with a total of one to ten thousand commits each, calculated over the full lifespan of the dataset. We argue that this sample of users adequately represents frequently active users without introducing much bias by bot-driven accounts that is expected to increase on more active accounts, given that ten thousand commits equates to almost four commits per day, for *every* day in the scope of the dataset. To nonetheless compare the findings, we also performed the analysis on the sample of users with between 10 and 100 thousand total commits. The expectation being, that the more commits a user has, the higher their activity density, and hence the higher the observed precision reduction effects. In total, we analysed 360 million commits by 160 thousand users in the range of one to ten thousand commits, and 100 million commits by 5000 users in the range of 10 to 100 thousand commits.

For each user, we counted the number of distinct activity points in time at various precision reduction levels from five minutes to one day. Activity points in time are given by the commit dates and are regarded as distinct, if—after applying the precision reduction—the remaining significant date information differs. The precision reduction is applied by rounding towards the next smaller integral multiple of the precision. As Fig. 4.5 shows, a 5 minute reduction level already results in only 75% distinct dates (median), and less than 50% at 1 hour. With the sample of very active users, we saw 54% and 28% for 5 minutes and 1 hour respectively. This indicates that a moderate precision reduction already prevents monitoring of intervals for a significant share of activities.

Functionality loss on the other hand is relevant to evaluate the cost of minimisation techniques. Such loss could be caused to the minimised application itself or to attached processes and workflows. As Git itself does not programmatically use commit dates but only passes them on, there is no direct loss of functionality or integrity. Usability should only be affected in the sense that users get unexpected results, e. g., for filtering, if they were unaware of the precision reduction. For instance, if a commit occurred within the last minute but was reduced to hour precision, a filter for *until 30 minutes ago* would list this command, provided no further precautions were taken. Such precautions could be to show a notice if filters conflict with timestamp precision or to reject them. The extent to which precision reduction affects Git workflows is of course very subjective. Our empirical data on chosen display and filter precisions is one indicator for reduction impact. Any reduction within the range of those commonly chosen precisions would have limited loss for workflows based on our analysed features. In qualitative interviews with four DevOps workers of different seniority, their stated interest in timestamp precision varied from no interest to precise oversight of team activities. This underlines our assumption that workflow-related interest in precise timestamps might be more driven by individual mannerism than procedural necessities. As user privacy should not be left to individual discretion, tools like Git should support to enforce the precision levels agreed upon on a per-team basis.

4.6.2 Representativeness and Limitations

Configurations on GitHub might not be representative for the overall user base of Git. Only users that desire a behaviour different from the default even make certain settings like aliases. However, a motivation to define aliases in general, is to make frequently used commands and arguments more easily accessible. Such settings are therefore not necessarily motivated by a wish to change default behaviour. We argue that the subset of users that define aliases is therefore not necessarily biased towards a date-related behaviour that differs from the default. The analysed settings might require a more experienced Git user to discover and use them. In that sense our analyses might be biased towards such users. To assess whether experience influences precision demand, future research could correlate our precision analysis with, e. g., commit counts. We argue that experience certainly factors into the discoverability of options, but presumably less into their configuration. Whether or not users need second-precision dates in a log output is likely unrelated to their experience.

4.6.3 Timestamp Reduction Approaches and Tools

Timestamp reduction could be applied on the presentation level, but to hinder performance monitoring and not be easily circumventable, it should be applied during recording. Wherever precision demand is highly user-specific, the recorded precision should be customisable. Nonetheless, a privacy-friendly default should be chosen that reflects most needs. If Git users wish to reduce the precision with which their actions are timestamped, they find no support to do so in Git today. And as dates are included in the input to the hash function that determines the commit hash, retroactive reductions interfere with hash chaining and history keeping. As such, modification to the dates might cause diverging Git histories. To nonetheless provide users with the option to reduce their timestamp precision, we built `git-privacy` [EMP], a tool that uses Git hooks to reduce timestamps while avoiding conflict with previously distributed states. It uses a unit annihilation approach similar to the rounding down described in Sect. 4.6.1, where users can choose the most significant time unit that should remain precise. In systems like Git with integrity-protected timestamps, at least excluding higher-precision timestamp parts from the integrity protection would allow post-recording reduction policies to take effect without compromising the history.

4.7 Summary and Conclusion

Using Git config files that users published on GitHub, we have compiled and analysed a large-scale dataset of features related to users' demand for timestamp precision. Our analysis of the usage of date and pretty formatting as well as date filters indicates that Git's current behaviour of recording dates to the precise second might not be justified by user demand. In fact, we found that when users customise output of subcommand aliases, over 40% of formats omit dates entirely. And of the remaining formats, 80% display dates with a reduced variable

or static day precision. As a result, a static full second precision is not utilised by nearly 90% of all subcommand pretty formats. Similarly, over 90% of date formatting in subcommand aliases uses variable or day precision. We saw a higher ratio of second precision output in pretty aliases as well as format and date settings, which could be due to users picking default date formats in pretty format stings, and due to a preference for ISO-style output. Our analysis of date filters found that only 0.5% of filters would require a precision of minute or second. In fact, 74% require a precision of day or less. All in all, we believe that our analysis provides strong empirical evidence, that user demand for precision can be met with less than second-precise timestamps. Our evaluation of possible privacy gains suggests that small precision reduction levels of a few minutes already have significant effects. As Git itself does not require any date precision, making it configurable would not only allow teams to define appropriate levels for their use case, but also facilitate a more GDPR-compliant use in companies. We encourage software engineers to employ reduced and adaptive precision timestamping for more proportionate solutions.

References

[BF19] Christian Burkert and Hannes Federrath. *"Towards Minimising Timestamp Usage In Application Software: A Case Study of the Mattermost Application"*. In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Vol. 11737. Lecture Notes in Computer Science. Springer International Publishing, 2019, pp. 138–155. ISBN: 978-3-030-31500-9. DOI: 10.1007/978-3-030-31500-9_9.

[Bur22] Christian Burkert. *".gitconfig Date Study Dataset"*. 2022. URL: https://github.com/EMPRI-DEVOPS/gitconfig-study-dataset.

[Cla+18] Maëlick Claes, Mika V. Mäntylä, Miikka Kuutila and Bram Adams. *"Do Programmers Work at Night or During the Weekend?"* In: ICSE. ACM, 2018. ISBN: 978-1-4503-5638-1. DOI: 10.1145/3180155.3180193.

[Dra+19] Kostas Drakonakis, Panagiotis Ilia, Sotiris Ioannidis and Jason Polakis. *"Please Forget Where I Was Last Summer: The Privacy Risks of Public Location (Meta)Data"*. In: *NDSS*. Jan. 2019. DOI: 10.14722/ndss.2019.23151.

[EMP] EMPRI-DEVOPS. *"git-privacy"*. URL: https://github.com/EMPRI-DEVOPS/git-privacy.

[ETL11] Jon Eyolfson, Lin Tan and Patrick Lam. *"Do Time of Day and Developer Experience Affect Commit Bugginess?"* In: *MSR*. ACM, 2011. ISBN: 978-1-4503-0574-7. DOI: 10.1145/1985441.1985464.

[Gam21] Game World Observer. *"Xsolla cites growth rate slowdown as reason for layoffs, CEO's tweet causes further controversy"*. 5th Aug. 2021. URL: https://gameworldobserver.com/?p=10949.

[Git21a] Git. *"Git Source Code"*. 2021. URL: https://github.com/git/git.

[Git21b] GitHub Docs. *"Best practices for integrators"*. 2021. URL: https://docs.github.com/en/rest/guides/best-practices-for-integrators.

[Git21c] GitHub Docs. *"Search API"*. 2021. URL: https://docs.github.com/en/rest/reference/search.

[Git22] Git. *"Reference"*. 2022. URL: https://git-scm.com/docs.

[Gou13] Georgios Gousios. *"The GHTorrent dataset and tool suite"*. In: MSR '13. 2013. ISBN: 978-1-4673-2936-1.

[MK18b] Paola Mavriki and Maria Karyda. *"Profiling with big data: Identifying privacy implications for idividuals, groups and society"*. In: *MCIS*. Oct. 2018.

[SA18] Awanthika Senarath and Nalin Asanka Gamagedara Arachchilage. *"Understanding Software Developers' Approach towards Implementing Data Minimization"*. 2018. DOI: 10.48550/ARXIV.1808.01479. URL: https://arxiv.org/abs/1808.01479.

[SLL06] Adam J. Slagell, Kiran Lakkaraju and Katherine Luo. *"FLAIM: A Multi-level Anonymization Framework for Computer and Network Logs"*. In: *LISA*. USENIX, 2006, pp. 63–77.

[TD18] Benjamin Traullé and Jean-Michel Dalle. *"The Evolution of Developer Work Rhythms: An Analysis Using Signal Processing Techniques"*. In: *Social Informatics*. Vol. 11185. 2018. DOI: 10.1007/978-3-030-01129-1_26.

[WZ19] Ian Wright and Albert Ziegler. *"The Standard Coder: A Machine Learning Approach to Measuring the Effort Required to Produce Source Code Change"*. In: *RAISE*. 2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE). May 2019. ISBN: 978-1-72812-272-4. DOI: 10.1109/RAISE.2019.00009.

III

Data Minimisation and Mitigation

5 PrivacyDates: A Framework for More Privacy-Preserving Timestamp Data Types

Citation	Christian Burkert, Jonathan Balack and Hannes Federrath. *"PrivacyDates: A Framework for More Privacy-Preserving Timestamp Data Types"*. In: Sicherheit 2022: Sicherheit, Schutz und Zuverlässigkeit: Konferenzband der 11. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI), 5.-8. April 2022 in Karlsruhe. Ed. by Christian Wressnegger, Delphine Reinhardt, Thomas Barber, Bernhard C. Witt, Daniel Christopher Arp and Zoltán Ádám Mann. GI-Edition. Proceedings Volume P-323. Gesellschaft für Informatik e.V. (GI), 2022. ISBN: 978-3-88579-717-3
Ranking	*n.a.*
Status	Published
Publication Type	Research Paper
Aim	The aim of this paper was to demonstrate and evaluate the practicality of alternatives to the conventional timestamp data types that were proposed in earlier work like the Mattermost case study [BF19].
Methodology	The paper presents a design based on timestamp alternative concepts found in the literature and conducts a design evaluation through a new case study with the Taiga application. The adapted design is implemented as a package for the Django Web Framework and again evaluated by integrating it in Taiga. This implementation evaluation includes an analysis of functionality as well as cost.

Continued on next page

Contribution	This paper demonstrates the practical applicability of less data-intensive alternatives to conventional timestamp data types. It presents a framework of alternatives that fits common conventional timestamp use cases and provides ready-to-use substitutions for timestamps with statically and dynamically reduced precision as well as context-aware counters. The paper demonstrates that an adoption of these alternatives is doable for large pre-existing projects even without prior knowledge of the application internals.
Co-authors' contribution	Jonathan Balack wrote the original bachelor's thesis (supervised by Christian Burkert) on which this paper is based. He performed the Taiga case study and drafted the initial timestamp framework, which was used as basis for the redesigned privacy dates framework presented in this paper. Hannes Federrath provided editorial guidance.

Abstract

Case studies of application software data models indicate that timestamps are excessively used in connection with user activity. This contradicts the principle of data minimisation which demands a limitation to data necessary for a given purpose. Prior work has also identified common purposes of timestamps that can be realised by more privacy-preserving alternatives like counters and dates with purpose-oriented precision. In this paper, we follow up by demonstrating the real-world applicability of those alternatives. We design and implement three timestamp alternatives for the popular web development framework Django and evaluate their practicality by replacing conventional timestamps in the project management application Taiga. We find that our alternatives could be adopted without impairing the functionality of Taiga.

5.1 Introduction

The design of software is today probably one of the biggest factors for everyday privacy. Since using software becomes virtually inescapable, it is increasingly application data modelling that decides how much of our personality and about our activities is recorded. Previous work [BF19] indicates that data models make excessive use of timestamps, the data type that adds the particularly sensitive temporal dimension to profiling. Timestamps have been previously observed to fulfil various functions in programming that not even require temporal properties. Instead, timestamps are frequently used for ordering or determining state (e. g., maintain order in which attachments were added). Function that can easily be achieved with less privacy-invasive alternatives. But also in cases where their temporal functions like universal comparability are actually used (e. g., time a bug report was filed), there appears to be room for a reduction of the typical second or even microsecond precision, to precisions that correspond more with human perception and increase privacy. Tackling the excessive use of timestamps in data models is a matter of raising awareness but also of providing ready to use alternatives. In this paper, we provide and evaluate a first such framework of timestamp alternatives. In summary, we make the following contributions:

- We design more privacy-preserving alternatives for common use cases of timestamp data types as identified by prior work.
- We validate the design applicability with a case study of the application *Taiga*.
- We provide an implementation for the popular web application framework Django.
- We evaluate and demonstrate the practicality of those alternatives by replacing timestamps in data model of Taiga with our alternatives and observe the effects.

The remainder of the paper is structured as follows: We firstly present related work and our adversary model, then we describe the design and implementation of our alternatives, after which we provide an evaluation.

5.2 Related Work

In a prior case study of the Mattermost application, we systematically analysed the usage of personally identifiable timestamps in data models [BF19]. We found that timestamps of creation, last modification and deletion are included in a majority of models. However, most user-related timestamps were found to have no programmatic use and only a small fraction is displayed on the user interface. Based on the identified functions of timestamps, we proposed design alternatives that use precision reduction and context-aware counters. Otherwise, the literature on timestamp-related privacy patterns and practical data minimisation is scarce. In 2017, a literature survey of privacy patterns by Lenhard et al. [LFH17] showed that proposals are rarely verified or even implemented. Strategies to reduce the sensitivity of timestamps have been proposed by Zhang et al. [ZBY06] for log sanitization. They discuss *time unit annihilation* as a strategy to gradually reduce precision over time.

5.3 Adversary Model

To contextualise privacy gain through our more data-minimal timestamp alternatives, we provide the following adversary model. It follows the established honest-but-curious (HBC) notion commonly used to assess communication protocols. Paverd et al. [PMB14] define an HBC adversary as a legitimate participant in a communication protocol, who will not deviate from the defined protocol but will attempt to learn all possible information from legitimately received messages. Following the adaption of this model to the context of application software and data models [BF19], we consider an adversary to be an entity that is in full technical and organisational control of at least one component of a software system, e.g., the application server. The adversary will not deviate from default software behaviour and its predefined configuration options, but will attempt to learn all possible information about its users from the data available in the application. This especially means that an adversary will not modify software to collect more or different data, or employ additional software to do so. However, an adversary can access all data items that are collected and recorded by the software system irrespective of their exposure via GUIs or APIs. We reason that this adversary model fits real world scenarios, because software operators in general lack the technical abilities to modify their software systems or are unwilling to do so, to not endanger the stability of their infrastructure or to not document potentially illegal behaviour. We come back to this adversary model when we employ server-side reduction later on.

5.4 Design

Based on alternative concepts for timestamps in the literature, we designed three data types: a generalised date with a static precision reduction (*rough date*), a context-aware counter for chronological ordering (*ordering date*), and a generalised date with temporally progressing

precision reduction (*vanishing date*). The designs are targeted as replacements for the conventional timestamp data type in the Django framework, but are using only standard database features typically available in development frameworks. The following describes the design for each alternative type.

5.4.1 Type 1: Rough Date

Rough date is a variation of Django's standard DateTimeField that truncates the date to a given precision. As such, it should offer all functionality that DateTimeField does and maintain the same interface, to be usable as a drop-in replacement. The desired precision is given as a mandatory argument at field initialisation, either in seconds or in the style of the timedelta class from Python's standard library package datetime [Pyt21] as multiples of the units minutes, hours, etc. The following creates a rough date with one hour precision: RoughDateField(hours=1). We deliberately do not provide a default precision to force users to consider the necessary precision for their given use case. The date part below a given precision is truncated.

5.4.2 Type 2: Ordering Date

The ordering date is an alternative to using timestamps for ordering items chronologically, if absolute date references and relative distances are not actually needed. OrderingDateField offers ordering via context-specific auto-incremented counters. Consequently, ordering date requires that objects are inserted in chronological order. As shown in Fig. 5.1, a context-defining string key is given for each OrderingDateField at model initialisation. The context label is cryptographically hashed to a 256 bit key which then uniquely identifies its corresponding OrderingContext which persists the actual counter state information. This way of maintaining the relation between ordering dates and their contexts in code and not in the database is space-efficient and allows for dynamic contexts keys. And since the context label is given at initialisation, ordering contexts can be defined very flexible. For instance, a label can comprise a username and thereby create an isolation between counter contexts of different users. This can be used to increase user privacy by avoiding an otherwise global counter context that would make instances with ordering date temporally chronologically comparable across users.

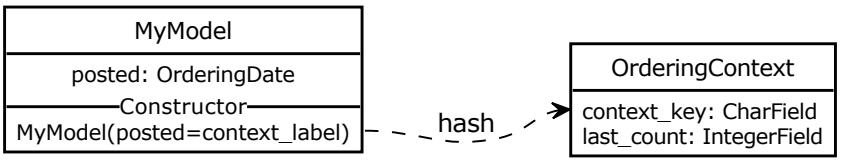


Figure 5.1. Class diagram showing a user-defined model with OrderingDateField. The related OrderingContext is identified by a context label given as initial field value. The integer value of OrderingDateField is then set to the context's next count.

5.4.3 Type 3: Vanishing Date

Vanishing date implements the privacy pattern of *time unit annihilation*. This alternative offers a progressing reduction of precision according to given increments until the end precision is reached. For each step, a precision is provided like for RoughDateField in combination with a temporal offset, i. e., the distance from object creation that marks when the reduction step is due. A background process regularly checks for due reductions and applies them. Listing 5.1 shows an example of a vanishing date with a three-step reduction policy, the first of which is immediately on creation, whereas the second and third follow after a given time. Table 5.2 lists the resulting stored date and next reduction event for each step.

Listing 5.1. Construction of a vanishing date with a three-step reduction policy ranging from initially 1 hour to finally 1 month precision after 7 days. Helper make_policy ensures correct reduction progression.

```
created_at = VanishingDateField(policy=make_policy([
  Precision(hours=1),
  Precision(days=1, after_hours=3),
  Precision(months=1, after_days=7),
]))
```

Step	Current Time*	Stored Date	Next Due Date
Creation/1st Red.	2021-11-08 15:17	2021-11-08 15:00	2021-11-08 18:00
2nd Reduction	2021-11-08 18:01	2021-11-08 00:00	2021-11-15 00:00
3rd Reduction	2021-11-15 00:03	2021-11-01 00:00	-

Table 5.2. Exemplary progression of vanishing date reduction with a 3-step policy leading to a precision of one month after seven days. (*Current times depend on the frequency and delay of periodic checks.)

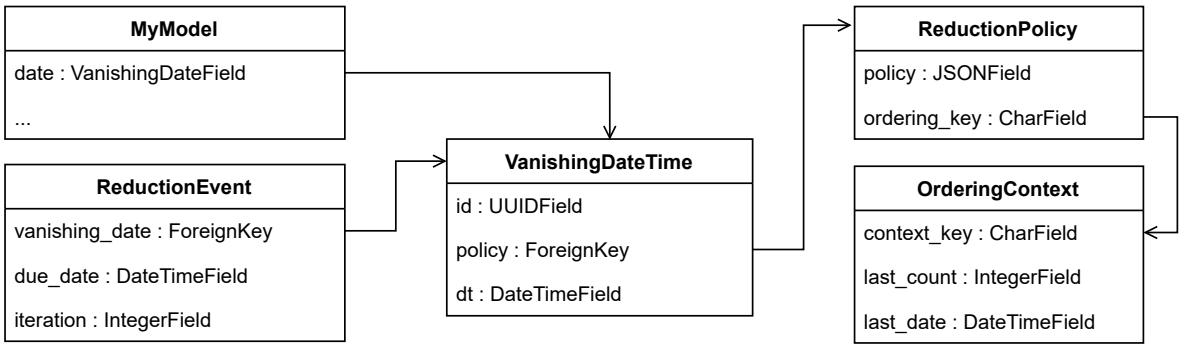


Figure 5.2. Class diagram showing a custom model that uses VanishingDateField which references a VanishingDateTime object specifying the information about the reduction policy and events.

As shown in Fig. 5.2, the resulting design of vanishing date is more complex than for rough date and requires auxiliary models to persist information about the reduction policy and the current progress within that policy. Therefore, VanishingDateField sets a reference to a VanishingDateTime class that holds the actual, gradually reduced date, a reference to a policy instance, and to a ReductionEvent that represents the next due reduction step. All instances of ReductionEvent form a queue that can be efficiently processed by the periodic due check. Since Django lacks the ability to natively trigger periodic tasks, we offer a management command for periodic reduction that can be triggered via, e. g., *Cron*.

Note that to not leave any traces of the previously truncated time information, due dates for subsequent reduction steps are calculated on the basis of the reduced step. In Table 5.2, for instance, the second reduction is due at 18:00 instead of 18:17, to not thwart the reduction to hour precision in the first step. As a result, the time periods given between each policy step are upper boundaries. In the previous example, the hour precision is available for 17 minutes less than the full three hours given in the policy. Also note, the reduction level of a step should not be larger than the offset of the following step.

Following our adversary model in Sect. 5.3, the application itself holds the only record of the timestamp reduced by vanishing date. This especially means, that the software operator does not keep a mirror of the information before reduction or of earlier reduction levels.

5.4.4 Design Validation

The purpose of design validation is to examine whether the design mainly based on previous case studies also holds for the application we selected to evaluate our implementation. As described below, we inspected its timestamp usage following the methodology of [BF19].

Taiga is a project management software that is built on the Django framework. Taiga focuses on user interaction like the creation, processing and commenting of tasks to control and document the progress of projects. Following the agile approach, interactions occur around planning elements like tasks, issues, epics, sprints and user stories. We chose Taiga for our evaluation because it is a popular app built on Django and it is focused on structuring and recording user interactions, which likely brings sufficiently complex requirements to test our implementation. In the following, we describe our methodology, the identified timestamp use, and any necessary modifications to our design.

Methodology

Following [BF19], we examine the source code of Taiga's back-end component [Tai21b] for occurrences of Django's date-related model fields DateTimeField, DateField, and TimeField. We assess semantic and purposes of each timestamp by examining all of their uses by the back-end, their presentation in the front-end [Tai21c] and their exposure via the API. We then use the identified purposes to select from our proposed types an alternative that provides the required

Model Field	Occurrences	Used in Models	Exposed via API
DateTimeField	170	48	41
DateField	15	3	3
TimeField	5	0	0

Table 5.3. Usage of date-related Django model fields in Taiga's back-end.

functionality. If none should be available, our design would need refinement. We find that the back-end REST API typically returns every attribute (model field) related to the requested object, regardless of whether the front-end uses them or not. Therefore, it is not sufficient to access timestamp usage and purpose only based on API exposure, but actual use based on inspections of the rendered front-end are necessary. To do so, we manually examined Taiga's front-end user interface cataloguing presented timestamp information. Usage in the back-end was assessed by manually inspecting all occurrences of date-related field names throughout the back-end code. These analyses were conducted on version 6.0.7 of both back-end and front-end, as released on March 8th, 2021.

Identified Timestamps

Table 5.3 shows the numbers of identified uses per model field. In total, we located 190 occurrences in Taiga's back-end code of which 51 are part of data model definitions. The remaining matches occurred in database migrations and serialization code. We did not further inspect the latter occurrences as they do not contribute any usage and purpose information. Almost all definitions use the DateTimeField. Only 3 (6%) use the DateField, whereas TimeField is not used in current model definitions at all.

To assess the API exposure, we inspected the source code and consulted the official API documentation for information about which fields are included in a query response. We found that all but 7 timestamps (86%) are exposed through the API. Of those 7 timestamps, 5 are also not programmatically used on the back-end. The visual inspection of the front-end UI also revealed that at least 22 (50%) of the timestamps fetched from the back-end API are not used there. Regarding those timestamps without a detectable usage or purpose, we can not determine a purpose-appropriate alternative. For the sake of data minimisation, they should be removed entirely. Also, not all timestamp fields are necessarily personal data. This is true for the three DateField uses, which are used to model due dates of planning elements (e. g., sprints) which are not directly linked to actions of users. Hence, we omit those from classification as well. For the remaining timestamps, we classified their type purpose according to their usage context in back-end and/or UI.

Timestamp Semantic and Purpose Classification

We classified the remaining timestamps based on the semantic given by their variable name and source code context. All models in Taiga (27) have a creation timestamp to automatically capture when a model instance was created. 17 models additionally record the time of the latest update, three the time a planning element was completed, and one when a notification was read. Regarding purpose classification, we followed [BF19] and used a bottom-up classification that inspects and labels each timestamp's programmatic use in the source code with respect the function they serve in the respective context, resulting into similar purposes: presentation for user information, sorting, and comparison. Table 5.5 in the appendix lists the identified purposes for each timestamp.

Design Revision

Based on the identified purposes and functional properties of our proposed alternatives, we select possible replacements for each timestamp. The selections are shown in Table 5.5 (appendix). If a timestamp is only used for sorting like the creation date of `Attachment`, the ordering date is the apparent alternative. Timestamps with a presentation or comparison purpose can equally be replaced by rough date and vanishing date. The latter should be chosen if an initial higher demand for precision exists.

We find that our proposed alternatives cover all found purposes. However, we noticed that the purpose of maintaining temporal order sometimes coincides with providing a temporal context (presentation or comparison). To replace such a timestamp, two fields are required in the initial design (e. g., OrderingDateField and VanishingDateField). Since this would both complicate usage and increase memory footprint, we decided to introduce two additional fields that combine the properties of ordering date with rough date and vanishing date respectively, which otherwise do not maintain order for dates reduced to the same value. To do so without increasing memory footprint, we use the sub-second value range available in most timestamp representations to hold the ordering counter. For a microsecond timestamp this leaves space for a 10^6 counter. The counter is incremented for all timestamps with identical values in their end-precision (Table 5.4). Note that this approach only works for timestamps that are added in chronological order, e. g., that are automatically set to the current time, otherwise the insertion order would not reflect their temporal order.

5.5 Implementation

We implemented our revised concept as a Django app that can be included into other Django projects to provide our date alternatives. It is available open source on GitHub [EMP22a]. In the following, we describe trade-offs and limitations of this implementation. Ordering date can simply build on available counter fields and is hence omitted from description.

Original	Vanishing 1. Iteration [5 sec]	Vanishing+Order 1. Iteration [5 sec]	Vanishing+Order 2. Iteration [30 sec]
12:20:11:673320	12:20:10:000000	12:20:10:000000	12:20:00:000000
12:20:14:313406	12:20:10:000000	12:20:10:000001	12:20:00:000001
12:20:17:248323	12:20:15:000000	**12:20:15:000002**	**12:20:00:000002**
12:20:33:040852	12:20:30:000000	12:20:30:000000	12:20:30:000000
12:20:35:917632	12:20:35:000000	12:20:35:000001	12:20:30:000001

Table 5.4. Sample sequence of vanishing date with and without added support to preserve ordering. The highlighted timestamps demonstrate that counter reset is determined by the end-precision of 30 sec which defines the counter scope.

5.5.1 Rough Date

To offer RoughDateField as a drop-in alternative for DateTimeField, it also has to support the options auto_now and auto_now_add, which automatically set the field to the current time at the moment when the object is saved (not initialised). To support these options, the reduction of precision has to be integrated in the saving process, since at any earlier point the value is not yet defined. To do so, we use pre-save hooks that apply the reduction. As a consequence, date values assigned to RoughDateField remain in full precision until saved.

5.5.2 Vanishing Date

As vanishing date is the most complex type, we face three main implementation challenges.

Avoiding chronology leak with UUIDs By default, Django would use an auto-incrementing integer primary key for VanishingDateTime if no other primary key was specified. Auto-incremented integers would however leak information about the temporal creation order of all instances of every model that uses VanishingDateField. For instance, an attacker could learn that user A logged in after user B posted their last comment but before user B closed the issue. To prevent such chronology leaks, we use randomized UUIDs as primary key. It should be noted that databases might still leak the temporal order by exposing the insertion order in certain queries. Future work should investigate options to, e. g., prevent users from executing such ordering queries.

Policy Reuseability As previously shown in Fig. 5.2, we decided to make ReductionPolicy a separate model to allow its reuse among vanishing dates with the same policy. We provide the helper function make_policy that transparently ensures policy reuse.

Model Identification with VanishingDateMixin An identification of all models that make use of VanishingDateField is required for two reasons: Firstly, to ensure a two-way cascading delete of VanishingDateTime, and secondly to create an initial ReductionEvent. To locate the relevant models, we decided to employ the common mix-in pattern, which uses inheritance to add functionality to a given class or model. Users of VanishingDateField have to add the VanishingDateMixin as a base class for their model. Thereby, we can automatically find all sub-classing models and register post-delete listeners to ensure a two-way deletion, as well as a post-safe listener to update reduction events.

5.6 Evaluation

In this section, we evaluate the practicality of our framework and its storage cost.

5.6.1 Practicality of Taiga Integration

To evaluate the practicality of our alternatives, we modified Taiga version 6.0.7 to use the timestamp replacements identified in Sect. 5.4.4 and detailed in the appendix. To test the correctness and impact of our modifications, we ran the modified Taiga and inspected the error output as well as the web front-end for potential negative effects. To check functional integrity, we created and modified various planning elements and compared the visible front-end behaviour before and after integrating our alternatives.

We found that all timestamps could be replaced as suggested, with few adjustments to the code base. As expected, rough date required the least effort of only replacing the field type. More effort was needed for the other alternatives: We used vanishing date and ordering date to each replace timestamps in three models. Since both replacements do not behave like a standard DateTimeField, all code that accessed or modified the field value had to be adjusted to either use the reference VanishingDateTime, or an appropriate context label instead. After these modifications, Taiga operated normally and we were able to create and modify elements as usual without functional impairments visible through UI or error log.

When it comes to presenting the replacements in the UI, rough date and vanishing date can mostly be treated like standard dates. However, to avoid confusion or wrong expectation of precision, the formatting of both should be adopted to reflect their precision. In contrast, ordering date can no longer be presented as a date in a meaningful way. But if the timestamp previously only fulfilled ordering purposes this should not be an issue. Otherwise, vanishing date might be a more fitting replacement.

5.6.2 Storage Cost

The following assesses storage cost compared to ordinary date and time (8 byte) using MariaDB as example [Mar19]. Rough date has the same memory footprint. Ordering date uses a 4 byte

counter reducing cost by half. The cost of vanishing date is dominated by three UUIDs, which require 38 byte. Added to two 8 byte date, one foreign key and one event counter (each 4 byte), a total of 138 byte is required for vanishing date, which is 17.25 times the cost of an ordinary date and time. Additionally, usage-dependent storage cost is added for vanishing date and ordering date by their auxiliary models. The number OrderingContext used depends on the number of distinct context labels set by developers, and scales with the number of users if individual contexts were used to avoid unnecessary comparability. In MariaDB, each OrderingContext requires 44 bytes. Moreover, a variable amount of storage is required for ReductionPolicy, which depends on the number of defined reduction steps. To give an overall example, applying the replacements in Table 5.5 increases Taiga's average storage cost per timestamp by about 5 times (not weighted by instance frequency).

5.7 Conclusion

Excessive, unthought use of timestamps in software data models is a violation of the data minimisation principle and potentially harmful to user privacy. Our case study of the Taiga application not only supports the findings of prior work regarding excessive use, but also in terms of minimisation potential through the use of more-privacy preserving timestamp alternatives. We have presented a framework of alternatives for common timestamp functions and purposes. We demonstrated its practicality by implementing it as a Django app which we then used to replace timestamps in Taiga. Although demonstrated for Django, these alternatives only use standard concepts and can be implemented for other development frameworks. Our evaluation with Taiga revealed that code changes were necessary but limited to adopting changed initialisation and access methods. Additionally, the presentation of timestamp may need adjustment to convey decreased precision levels. Depending on the selection of alternatives, especially the frequency of vanishing date, storage cost might increase noticeably. Where this is not acceptable, rough date and ordering date can be used with little to no additional storage cost, but without gradual reduction. Our integration test suggests that more privacy-preserving alternatives can be adopted with reasonably low effort. A user study to evaluate their usability with developers is left to future work.

Acknowledgements The work is supported by the German Federal Ministry of Education and Research (BMBF) as part of the project Employee Privacy in Development and Operations (EMPRI-DEVOPS) under grant 16KIS0922K.

Appendix: Taiga Timestamp Purposes and Replacements

Model / Timestamp	Presentation	Sorting	Comparison	Replacement
Attachment				
created_date		✔		OD
Epic				
created_date	✔			RD
HistoryChangeNotification				
updated_datetime			✔	RD
HistoryEntry				
created_at	✔	✔		VD+O
delete_comment_date	✔			VD
edit_comment_date	✔			VD
Issue				
created_date	✔			RD
modified_date	✔	✔		RD
Like				
created_date			✔	RD
Task				
created_date	✔			RD
TimeLine				
created	✔	✔	✔	VD+O
User				
date_joined	✔			RD
UserStory				
created_date	✔			RD
Watched				
created_date		✔		OD
WebNotification				
created	✔	✔		VD+O
read		✔		OD
WikiPage				
created_date	✔			RD
modified_date	✔			RD

Rough Date **(RD)** Ordering Date **(OD)** Vanishing Date **(VD)**
Vanishing Date with Ordering **(VD+O)**

Table 5.5. Identified purposes and suggested replacements for used timestamps in Taiga.

References

[BF19] Christian Burkert and Hannes Federrath. *"Towards Minimising Timestamp Usage In Application Software: A Case Study of the Mattermost Application"*. In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Vol. 11737. Lecture Notes in Computer Science. Springer International Publishing, 2019, pp. 138–155. ISBN: 978-3-030-31500-9. DOI: 10.1007/978-3-030-31500-9_9.

[EMP22a] EMPRI-DEVOPS. *"django-privacydates"*. 2022. URL: https://github.com/EMPRI-DEVOPS/django-privacydates.

[LFH17] Jörg Lenhard, Lothar Fritsch and Sebastian Herold. *"A Literature Study on Privacy Patterns Research"*. In: *43rd Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2017, Vienna, Austria, August 30 - Sept. 1, 2017*. IEEE Computer Society, 2017, pp. 194–201. DOI: 10.1109/SEAA.2017.28.

[Mar19] MariaDB. *"Data Type Storage Requirements"*. Apr. 2019. URL: https://mariadb.com/kb/en/data-type-storage-requirements/.

[PMB14] AJ Paverd, Andrew Martin and Ian Brown. *"Modelling and automatically analysing privacy properties for honest-but-curious adversaries"*. Tech. rep. 2014.

[Pyt21] Python Software Foundation. *"datetime - Basic date and time types"*. May 2021. URL: https://docs.python.org/3/library/datetime.html.

[Tai21b] Taiga Agile. *"taiga-back"*. Mar. 2021. URL: https://github.com/taigaio/taiga-back/releases/tag/6.0.7.

[Tai21c] Taiga Agile. *"taiga-front"*. Mar. 2021. URL: https://github.com/taigaio/taiga-front/releases/tag/6.0.7.

[ZBY06] Jianqing Zhang, Nikita Borisov and William Yurcik. *"Outsourcing Security Analysis with Anonymized Logs"*. In: *Second International Conference on Security and Privacy in Communication Networks and the Workshops, SecureComm 2006, Baltimore, MD, USA, August 2, 2006 - September 1, 2006*. IEEE, 2006, pp. 1–9. DOI: 10.1109/SECCOMW.2006.359577.

6 Analysing Leakage during VPN Establishment in Public Wi-Fi Networks

Citation	Christian Burkert, Johanna Ansohn McDougall, Hannes Federrath and Mathias Fischer. *"Analysing Leakage during VPN Establishment in Public Wi-Fi Networks"*. In: *ICC 2021 - IEEE International Conference on Communications*. ICC 2021 - IEEE International Conference on Communications. Montreal, QC, Canada: IEEE, June 2021, pp. 1–6. ISBN: 978-1-72817-122-7. DOI: 10.1109/ICC42927.2021.9500375
Ranking	CORE 2018 B ERA 2010 B QUALIS 2016 A1
Status	Published
Publication Type	Research Paper
Aim	The aim of this paper was to analyse if and how mobile workers are susceptible to traffic leakage in public Wi-Fi networks even if they make use of VPN apps as a protective measure. This was motivated by our hypothesis that VPN tools do not sufficiently protect against leakage *during* the establishment of the tunnel and that operating systems and VPN apps do not sufficiently cooperate to handle captive network environments without deadlocks or leaks.
Methodology	We use a series of lab experiments where we confront various VPN apps on different popular operating systems (including mobile) with a self-operated captive Wi-Fi network and capture the device's traffic on the access point, to detect and classify all traffic that is not part of the captive network remediation or VPN establishment and thus can be considered a leak. We use a constructive approach to develop a leak-free connection process based on the observed failures.

Continued on next page

Contribution	This paper is the first do systematise the state of VPN APIs for popular operating systems and conduct an experimental evaluation of VPN app behaviour with regard to traffic leakage in captive network environments. We found that a large majority of VPN apps are currently unable to provide VPN establishment without leaking traffic or causing deadlocks, and that operating systems currently fail to provide APIs that sufficiently support a leak-free implementation. The paper also proposes a leak-free process that coordinates connection startup between operating system and VPN app.
Co-authors' contribution	The article was co-authored by Johanna Ansohn McDougall, who assisted with developing the experimental methodology, visualisation and editing, as well as Hannes Federrath and Mathias Fischer, who both provided editorial guidance.

Abstract

The use of public Wi-Fi networks can reveal sensitive data to both operators and bystanders. A VPN can prevent this. However, a machine that initiates a connection to a VPN server might already leak sensitive data before the VPN tunnel is fully established. Furthermore, it might not be immediately possible to establish a VPN connection if the network requires authentication via a captive portal, thus increasing the leakage potential. In this paper we examine both issues. For that, we analyse the behaviour of native and third-party VPN clients on various platforms, and introduce a new method called *selective VPN bypassing* to avoid captive portal deadlocks.

6.1 Introduction

Public Wi-Fis supply Internet connectivity on the go. However, their usage comes with considerable privacy risks: A Wi-Fi operator can monitor all traffic, analyse the metadata and, in case of unencrypted connections, even expose its users to nearby sniffers and attackers [Che+13; Som+19]. VPNs are used to mitigate these dangers by applying an additional layer of encryption. However, they can also give a false sense of security: Leakage of traffic can already occur while a user attempts to connect to a VPN, and a captive portal might even force the user to temporarily disable the VPN altogether, because—as we will show in this paper—many VPN clients interfere with captive portal detection. After joining the network, running applications like mail or chat clients will themselves attempt to connect to their servers. During the time the VPN is not yet established, this might leak potentially sensitive information about a user's habits, preferences, or work environment to the network.

VPNs were originally designed to establish connectivity to remote private networks and to access their remote services. Nowadays, they are mostly used for privacy-friendly surfing: They aim at masking the original source IP addresses with that of the VPN endpoint and thereby protecting their users, e. g., from observation by Internet Service Providers (ISPs). For that use case, it is crucial that *all* traffic is routed via the VPN tunnel and nothing is leaked to the intermediate network besides the VPN connection itself.

With this paper, we are the first to examine the issue of secure VPN establishment in captive networks and present evidence that native VPN clients shipped with Windows, macOS, iOS, Android, and Ubuntu/GNU Linux, as well as popular third-party clients fall short in protecting user privacy during the establishment of VPN connections. To summarise, we make the following contributions:

- We systematise the current state of system APIs for VPNs.
- We analyse OS mechanisms for captive portal detection.
- We examine the behaviour and leakage of native and third-party VPN clients, including in captive networks.

- We introduce *Selective VPN Bypassing*, a concept of gradual and selective network capability management to avoid leakage during captive network remediation and VPN connection establishment.

The remainder of this paper is structured as follows: In Sect. 6.2, we provide background and terminology. Sect. 6.3 presents related work. In Sect. 6.4, we define the requirements for a secure VPN establishment. Sect. 6.5 describes the status quo on VPN APIs. In Sect. 6.6, we analyse VPN clients and APIs for violations of our security requirements. Sect. 6.7 proposes a design for a leak-free VPN establishment, before Sect. 6.8 concludes the paper.

6.2 Background and Terminology

In this section, we briefly describe key concepts of Wi-Fi communication, captive portals and VPNs, and introduce additional terminology used throughout the paper.

A *Public Wi-Fi* or *Hotspot* is an 802.11 Wi-Fi network that is open to the public, i. e., accepts connections from any client. Unless explicitly stated, we assume public Wi-Fis to operate without encryption. To mitigate the potential dangers of surfing in an unencrypted public Wi-Fi, users can decide to increase their security by utilising a VPN. With respect to VPNs, we introduce the term *VPN Bootstrapping*: It describes the process of blocking all traffic except that required to establish the VPN connection until the VPN tunnel is successfully established.

While public Wi-Fis can often be used without special access rights, providers can present their customers with a *Captive Portal* (CP): A website that users are automatically redirected to that contains terms of service and sometimes the necessity to input credentials. Until the terms of the captive portal are fulfilled, access to the Internet is blocked. The process of signing-in and lifting the network block is denoted as *remediation*. We use the term *Captive Network* (CN) to refer to hotspots containing a captive portal. CPs can be explicitly announced via a DHCP option or a Router Advertisement (RA) extension, which informs the client of the URI needed to access the authentication page. While these announcement options exist and have been standardised [Kum+15], they are not widely adopted in practice. Instead, platforms apply heuristics to detect captive networks: Upon successfully connecting to a network, clients send out HTTP requests to a predefined URL, expecting a predefined response, e. g., an HTTP status code 204. A CN instead replies with an HTTP redirect (e. g., status code 307), redirecting the user to the CP [Wik20]. Thereby, the OS assumes a CN and displays the CP.

When attempting to use a VPN in a CN, a *Captive Deadlock* can occur: in it, the leak prevention of a VPN client blocks the communication with the CP that is necessary to gain an Internet uplink, and thereby indirectly also blocks the route to the VPN endpoint.

6.3 Related Work

The security and privacy of public Wi-Fis and VPN client software has been extensively studied in the literature. [Ali+19; Che+13; Som+19] examine risks caused by public Wi-Fi and captive portals, and the reason why people use them nonetheless. [Ikr+16] and [WMB20] analyse the VPN clients on mobile platforms. [Ikr+16; Kha+18] and [Per+15] verify the security and privacy claims of commercial VPN clients. Among other things, they discover severe leakage of IPv6 and DNS traffic: Up to 84% of VPN apps don't tunnel IPv6 [Ikr+16], and around 60% of VPN apps use Google's DNS servers, while only about 10% use own DNS resolvers [Ikr+16].

However, regarding traffic leakage during VPN connection establishment, there is very little prior work and—to the best of our knowledge—we are the first to analyse VPN clients and their behaviour in captive networks. Karlsson et al. [Kar17] present a prototypical device that connects to public Wi-Fis, opens up a VPN tunnel and then creates an encrypted Wi-Fi for the user to connect to, such that all traffic is routed through the VPN tunnel on the intermediate device. This mitigates the startup leakage issue by moving it from the user device to the intermediate device which presumably exposes less sensitive traffic of its own. However, we argue that the requirement to maintain an additional device is impractical to most users.

6.4 Requirements for Secure VPN Bootstrapping

To ensure a secure and privacy-preserving establishment of VPN connections in public Wi-Fis, we propose the following requirements for secure VPN bootstrapping:

R1: Always-on Functionality: A client offers an always-on or similar functionality that asserts that a VPN tunnel is established when network connectivity is available. If not automated, the user must be able to activate this functionality without an existing Internet connection, e. g., before joining a public Wi-Fi.

R2: Captive Network Support: The VPN client allows the OS to perform captive portal detection and remediation or performs it itself. The client does not cause a captive deadlock.

R3: Minimal startup traffic: No traffic is sent from the client device that is not necessary to remediate a captive network or to establish a VPN tunnel.

R4: Blocking Fail State: Outbound traffic continues to be blocked if a VPN tunnel cannot be successfully established (e. g., if the VPN endpoint is unreachable).

R5: No Tunnel Bypass: After successful VPN tunnel establishment, no non-VPN traffic, such as previously started TCP streams, bypass the tunnel. Instead, any preexisting connection is interrupted and reestablished through the tunnel. Periodic requests to check the state of the captive network are exempted.

6.5 VPN API Status Quo

In this section, we describe the current state of system APIs available for VPNs on major platforms according to developer documentation.

Apple macOS and iOS As part of their network extension framework, Apple offers an API for creating VPN apps that build on Apple's system VPN functionality (Personal VPN [Appc]) or provide custom protocol implementations (Packet Tunnel Provider [Appb]). This API offers always-on functionality (R1) via so called on-demand rules, which can be configured to trigger, e. g., when a Wi-Fi connection is established [Appd]. According to the documentation, such on-demand connection rules block outgoing traffic until the VPN tunnel is established (R3).

Android Developers can build VPN apps using the system API and the `BIND_VPN_SERVICE` permission. VPN apps can run in, among others, always-on (R1) and per-app mode. Always-on VPN connections are kept alive unconditionally by the system as long as the device is running and Internet connectivity is available. Developers of VPN apps can specify lists of allowed and disallowed apps whose traffic is to be tunnelled through the VPN. It is also possible to block all connections outside the VPN tunnel, which results in disallowed apps losing all network connection [And].

Windows 10 Always-on functionality (R1) is built in as an auto-trigger for VPN profiles.[1] In general, VPN profiles can be provided by a VPN app[2] or via a mobile device management mechanism to remote-join clients to a domain[3].

Ubuntu GNU/Linux Since the landscape of GNU/Linux distributions is very diverse, we focused our analysis on the popular desktop distribution Ubuntu. Ubuntu uses NetworkManager (NM) as its high-level daemon for networking including VPN. NM provides APIs to plug in VPN services via DBus[4] and libnm[5]. VPN services can declare themselves persistent, i. e., they will attempt to maintain the connection across link changes and outages.[6] VPNs can be further set as so-called *secondaries* for other connections to be activated in reaction to the other connection going online (R1).

[1] https://docs.microsoft.com/en-us/windows/security/identity-protection/vpn/vpn-auto-trigger-profile
[2] https://docs.microsoft.com/en-us/uwp/api/windows.networking.vpn.ivpnprofile?view=winrt-19041
[3] https://docs.microsoft.com/en-us/windows-server/remote/remote-access/vpn/always-on-vpn/deploy/always-on-vpn-deploy
[4] https://developer.gnome.org/NetworkManager/stable/gdbus-org.freedesktop.NetworkManager.VPN.Plugin.html
[5] https://developer.gnome.org/libnm/stable/NMVpnServicePlugin.html
[6] https://developer.gnome.org/libnm/stable/NMSettingVpn.html

6.6 Experimental Analysis

While the existing VPN APIs partially contain mechanisms to fulfil the requirements R1 and R3, apps using the API don't necessarily utilise the functionality or fail to fulfil the other requirements. In this section, we present an experimental analysis concerning OS-specific Captive Portal Detection (CPD) mechanisms, and then test different VPN clients with respect to their fulfilment of the requirements.

6.6.1 Testbed Setup and General Procedure

Setup

Our testbed comprises a Raspberry Pi that runs the Wi-Fi Access Point (AP) and also captures the incoming Wi-Fi traffic of our test clients. We use a Raspberry Pi 3 Model B+ running Raspbian GNU/Linux 10 and with hostapd 2.2.7 providing the WPA2-secured AP and Nodogsplash 4.5.1 beta providing the captive portal functionality. The captive network is set up to redirect (status code 307) plain HTTP requests to the captive portal sign-in page running on the same Raspberry Pi, where the user can gain Internet access by clicking a continue button. The traffic capturing is done with Wireshark/tshark.

Regarding the test clients, we used the following setup: A Google Nexus 5X running Android 10.0[7], an iPad running iOS 13.7, a MacBook Pro running macOS 10.15.7, Ubuntu 20.04 LTS with NetworkManager 1.22.10, and Windows Education 10.0.19041 both running on a Dell laptop.

Leak Classification

We consider all outgoing traffic that is not required for VPN setup as leakage. However, the low-level protocols ARP, EAPOL, DHCP, ICMP, IGMP, LLMNR, and mDNS are not considered leakage. We further distinguish leakage to hosts operated by the platform maintainer (e. g., Apple for iOS and macOS, or Microsoft for Windows) from leakage that is going to other third-party hosts.

General Test Procedure

We test each VPN client in a captive network (denoted as *captive mode*) and in an unrestricted network (*open mode*) as well as in a network that selectively but permanently blocks, i. e. drops, the traffic to the respective VPN endpoint (*block mode*). Note that in block mode, both the safe fail state requirement (R4) is tested, as well as the probability to detect potential race conditions

[7] Using PixelExperience ROM version 10.0-20200912-1735

	macOS	iOS	Windows	Android	Ubuntu
System employs CPD	✓	✓	✓	✓	✓
Blocking of platform traffic	✓	✗	✗	✗	✗
Blocking of third-party traffic	✓	✓	✗	✓	✗

Table 6.2. Overview of platform behaviour during captive network remediation.

during the VPN startup is increased. If a client does not offer auto-connect functionality, we instead manually activate the connection before joining the testbed Wi-Fi.

Each network configuration is tested and captured at least three times while performing the following steps: 1) disconnect client device from Wi-Fi, 2) activate VPN client if necessary, 3) clear CP state and start capture, 4) connect client to Wi-Fi, 5) complete CP sign-in if prompted, and 6) continue capture for 20 seconds. Afterwards, we inspect the captured traffic and classify it according to our leakage definition.

6.6.2 Captive Portal Detection Mechanisms

To establish a baseline, we analysed each platform's behaviour when confronted with a captive network. These tests were conducted without any VPN enabled. We analysed two scenarios: a) CP sign-in is completed, i. e., the continue button on the CP website is clicked, and b) CP sign-in is omitted, leaving the client captured. We observed that all platforms employ effective Captive Portal Detection (CPD) mechanisms and prompt the user to sign in to the captive network. However, the platforms exhibit the following differing behaviour regarding leakage during CPD, which is also summarised in Table 6.2: On macOS and iOS, we find that CPD takes place before connectivity is available to the rest of the system. If the sign-in is omitted, macOS and iOS will continue to block all other outgoing traffic. However on iOS, DNS queries were leaked about non-CPD-related platform hostnames, followed by IP traffic to those hosts. On Android, DNS lookups as well as TCP traffic to non-CPD Google hosts take place before remediation. On Ubuntu, the CPD appears non-blocking. We observed outbound traffic to third-party global and local destinations. On Windows, we observed non-CPD traffic to Microsoft hosts as well as to third-party hosts.

In the following sections, we present the findings of our analysis of the native VPN clients, the macOS API and third-party VPN clients. The results are summarised in Table 6.3 and will be discussed in Sect. 6.6.6.

6.6.3 Native VPN Clients

All tested operating systems ship with built-in clients that offer basic VPN functionality. Note that this analysis only covers the functionality that is exposed by the OS's native GUI (e. g., via

Platform	Client	R1: Always-on	R2: CPD	R3: Minimal	R4: Blocking	R5: No Bypass
macOS	Native	✗	–	–	–	–
	Demo	✓	✓	✗	✗	✗
	EncryptMe	✓	✓	✗	✗	✗
	ExpressVPN	✓	✗	(✓)	✓	?
	Mullvad	✓	✗	✓	✓	✓
	ProtonVPN	✓	✓	✗	✓	✗
iOS	Native	✗	–	–	–	–
	EncryptMe	✓	✓	✳	✓	✗
	ExpressVPN	✓	✓	✳	?	✗
	Mullvad	✓	✓	✓	✓	✓
	ProtonVPN	✓	✓	✳	✳	✗
Windows	Native	✗	(✗)	–	–	–
	EncryptMe	✓	✓	✗	✗	✗
	ExpressVPN	✓	✓	✗	✗	✓
	Mullvad	✓	✗	✓	✓	✓
	ProtonVPN	✓	✗	✓	✓	✓
Android	Native	✓	✗	✓	✓	✓
	EncryptMe	✓	➠	✗	✗	✗
	ExpressVPN	✓	✗	✳	✓	✓
	Mullvad	✓	✗	✳	✓	✓
	ProtonVPN	✓	➠/✗	✗/✓	✗/✓	✳/–
Ubuntu	Native	✓	✗	✗	✗	✓
	ExpressVPN	✓	✗	✗	?	✓
	Mullvad	✓	✗	✓	✓	✓

Table 6.3. Overview of the VPN client analysis. Symbol usage: race condition (➠), platform traffic leak (✳), not testable (**?**), not applicable (–)

a network settings dialogue) and not functionality that is only exposed via APIs or configuration files.

macOS and iOS

Although provided by system APIs (cf. Sect. 6.5), always-on functionality is neither part of the native VPN client on macOS (version 10.15.7) nor iOS (version 14.0.1). VPNs set up via the on-board configurators have to be started manually while Internet connectivity is available and do not always (re-)connect if network connectivity is interrupted. Note that iOS additionally supports remotely deploying always-on VPN profiles via device supervision, i. e. mobile device

management.[8] As we do not have the necessary prerequisites for device supervision, this option was not included in our analysis.

Android

The built-in VPN client [And] offers the option to turn on *always-on* VPN if the VPN server address is provided numerically and a DNS server is set. We found that with always-on enabled, Android entered a captive deadlock. Disabling the captive network allowed Android to establish the VPN tunnel. We found that apart from the CPD traffic, TLS traffic was sent to `www.google.com` before the tunnel was established. Otherwise, no further traffic was leaked.

Ubuntu GNU/Linux

Ubuntu ships with NM as a VPN and networking client. Provided that a VPN profile is configured, NM allows to set an always-on VPN profile per connection (e. g., an SSID) in the connection editor. In captive mode and with auto-connect enabled, NM stops forwarding CPD requests, thus causing a captive deadlock. In open mode, the VPN starts as expected. In any case, NM's auto-connect has no blocking effect and outbound traffic is unrestricted, both during establishment and if the VPN is unreachable. In fact, NM appears to react to ICMP notifications stating that the VPN destination is unreachable by setting the connectivity to offline, thus preventing further leaks. However, during the time NM attempts to connect to the VPN, no restrictions are in place for other process to send traffic. In block mode, when no ICMP notification is received because the VPN traffic is dropped, NM retains online connectivity until the VPN times out, thus increasing the time for other processes to leak traffic.

Windows

Windows 10's built-in graphical VPN configurator is not able to set up an always-on VPN profile. However, we found that VPN connections are not immediately disconnected if a network interface goes offline. If the network goes online again before the VPN connection times out, we observed that CPD is suppressed and the system is in a deadlock until the VPN timeout occurs, after which outbound traffic is unrestricted. Note, that before the timeout and in open mode, we also observed VPN bypasses by DNS lookups for Microsoft hosts like `www.bing.com`.

[8] https://support.apple.com/en-gb/guide/deployment-reference-ios/iore8b083096/1/web/1

6.6.4 VPN API Demo

Platforms like macOS and iOS provide dedicated APIs that supposedly offer blocking VPN bootstrapping. However, as they currently don't integrate this functionality in their native VPN clients, we implemented a minimal demo for macOS to test the validity of the documented properties. We selected macOS for our demo because of previous development experience on that platform.

The demo builds on the Personal VPN API (cf. Sect. 6.5) and simply registers an on-demand VPN that auto-connects with an `NEOnDemandRuleConnect` rule every time a Wi-Fi connection is used. While testing our demo, we found that the VPN tunnel is reliably started by the system after we connect to our test Wi-Fi. However, our traffic captures consistently show outgoing platform (towards Apple) and third-party traffic between the CP authentication and the tunnel establishment in both open and captive mode. The documented blocking feature of the `NEOnDemandRuleConnect` rule appears to be insufficient. We further found that after the tunnel establishment, TCP traffic bypasses the tunnel that originates from before the VPN establishment, i. e., for ongoing TCP streams. We also observed ongoing TCP re-transmission attempts that started before the establishment. This corroborates a bypassing vulnerability that has been reported in March 2020 but remains unfixed (in Oct. 2020) [Pro20]. In block mode, we observed continued traffic to platform and third-party hosts after the failed VPN connection attempts. The on-demand connection rule appears not to sufficiently block on a failed establishment.

6.6.5 Third-Party VPN Clients

We additionally included the following third-party clients in our analysis: ExpressVPN as the self-declared market leader, EncryptMe as a benchmark used by [Kar17], ProtonVPN as Top 10 provider with open-source clients and an active stand in VPN leak prevention [Pro20], and Mullvad VPN as an open-source provider.

ExpressVPN

On Ubuntu, version 3.0.2.12 deadlocked in captive mode. In open mode, we observed no leaks besides local traffic. Block mode tests were not practical because the endpoint address switched unpredictably. We further found that when the app enters a blocking 'unable to connect' state, it could only be re-established with full leakage, despite using the connect statement as described in the info text, which apparently temporarily disables the traffic block.

On iOS, we tested version 8.3.5 which uses the on-demand API to automatically reconnect the tunnel once it is activated. We found non-CPD platform traffic during CPD in captive and open mode. Additionally, we saw TCP resets of previous platform connections bypassing the tunnel. No deadlocks occurred. Block testing was skipped again.

On macOS, version 7.11.6(6), according to network settings, does not use Apple's on-demand API. In captive mode, we observed leak-free deadlocks as the CPD is interrupted. In open mode, the CPD passes but the tunnel establishment repeatedly fails to complete within the capture time. However, within that time we found no leaks. In block mode, ExpressVPN holds outgoing traffic and prevents leaks.

On Windows, we tested version 9.1.0(258). In captive and open mode, we saw non-CPD platform and third-party traffic before remediation and tunnel establishment. In block mode, no additional leakage occurs. The windows client appears to not sufficiently block startup leaks.

On Android, version 9.0.40 deadlocked in captive mode without leaks except non-CPD platform traffic to `www.google.com`. In open mode, we saw the same leaks but no deadlocks. Block mode caused no additional leaks.

EncryptMe

This client has been used as a benchmark by [Kar17]. It offers VPN-activation based on network trustworthiness. On macOS, we tested version 4.2.3 with activated auto-start and leak protection (OverCloak). We found platform and third-party traffic leaks in open, captive and block mode before VPN startup and afterwards. No deadlocks occur.

On iOS, we analysed version 4.4.4 with enabled auto-protect which uses iOS's on-demand functionality. In open mode, the app shows non-CPD platform leaks. During captive mode, we also found continued platform traffic bypassing the VPN. In block mode, we saw no additional leakage.

On Windows, we analysed version 1.1.0. In captive mode, we observed third-party leaks and no deadlocks. In open mode, the VPN establishment completes faster, hence we found fewer and only platform leaks. However, blocking VPN traffic causes third-party traffic to surface again.

On Android, version 4.2.0.1.81964 exhibits indeterministic captive deadlocking and third-party leaks, indicating a race condition between the CPD and the tunnel establishment. After blocking the VPN in captive mode, we observed consistent leak-free deadlocks. In open mode, we found no leaks. However, blocking the VPN in open mode causes third-party traffic to leak. A Linux version is not available.

Mullvad VPN

On iOS, version 2020.4 established a tunnel after remediation without leaks. A source code inspection confirms that Mullvad uses the on-demand connection API. Open or block mode cause no leaks either.

On macOS, Windows and Ubuntu, we tested the respective client in version 2020.5 and observed the same behaviour: In captive mode, the clients caused a leak-free deadlock by

suppressing the CPD traffic. In open and block mode, the client prevents leakage and we observed only VPN traffic.

On Android, Mullvad is still in beta version (2020.6-beta2). In captive mode, CPD requests are sent and redirected, but the request to the CP is lost, thus causing a deadlock. In open mode, we observed non-CPD platform traffic before tunnel establishment. Block mode triggers no additional leaks.

ProtonVPN

On iOS, version 2.2.4 offers always-on functionality that cannot be turned off. We observed non-CPD platform DNS and IP traffic between remediation and tunnel establishment. We also found traffic to the global IP address of our testbed gateway throughout the capture. Blocking the VPN traffic exhibited the same leakage. We additionally observed traffic to Akamai servers, presumably operating for Apple.

On macOS in captive and open mode, ProtonVPN 1.7.2 exhibited no leaks during remediation, but we subsequently observed non-CPD platform and third-party DNS and IP traffic before tunnel establishment. After tunnel establishment, prior platform and third-party TCP streams and local traffic continued. We also noted reverse DNS lookups of the test client's local IP address. In block mode, we saw the same leakage up to the point of the attempted tunnel establishment. After that, no further leaks were visible.

On Windows, version 1.17.3 does not connect automatically, but can be started manually before connecting to the Wi-Fi. In captive mode, no CPD requests are sent. The system is in captive deadlock. In open and block mode, we saw no leaks but traffic to `api.protonvpn.ch`.

On Android, we tested version 2.3.54.0. In captive mode, we observed inconsistent behaviour: the CP request is either suppressed and the system deadlocked, or sent before tunnel establishment alongside other platform traffic. This behaviour indicates a race condition in the CPD and VPN handling. In block mode, increased platform and third-party leaks appear. In open mode, we observed platform traffic leaks throughout. In its settings, ProtonVPN for Android contains instructions to activate always-on functionality by manually enabling Android's always-on and blocking VPN properties in the system VPN profile. With always-on enabled, captive and block mode lead to a leak-free deadlock. Because the app explicitly guides the user to that system feature, we also included these results.

Note that ProtonVPN also offers a command-line client for Linux, however we were unable to test it, because our credentials were rejected.

6.6.6 Summary and Discussion

Of the native and third-party VPN clients in our analysis (see Table 6.3), only Mullvad for iOS managed to establish a connection in a captive network without deadlocks or traffic leaks. On

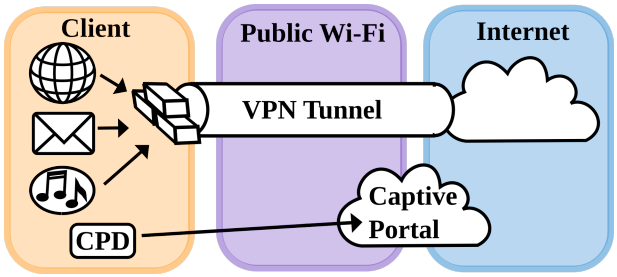


Figure 6.1. Stage 1 selective VPN bypass: only CPD traffic is allowed

all platforms but iOS and macOS, an effective leak protection coincided with deadlocks after a failed CPD. We identified issues with violations of the minimal startup requirement (R3) due to platform traffic leaks on Android and iOS, that appear to be related to system APIs making exceptions for platform destinations in an otherwise effective leak prevention (esp. on iOS). The appearance of race conditions on Android suggests that APIs do not sufficiently ensure a prioritised VPN startup or guide developers towards secure API usage. We found and reported several issues to Apple, Google, and GNOME, including third-party leaks during macOS's supposedly blocking on-demand connection handling. We are continuing to report and discuss found issues with the other vendors.

Based on our analysis, we are convinced that a secure and private VPN establishment, esp. in public Wi-Fi environments, relies on a deep integration with the OS and its CPD process. It therefore requires high-level system APIs that allow VPN clients to hook themselves into a multistage secure network startup process, which we detail in the following.

6.7 Selective VPN bypass for Captive Portals

We propose the following design for a secure VPN startup implementation. The process has to selectively allow outgoing traffic in three stages:

Stage 1: Captive Portal Detection In this stage, as depicted in Fig. 6.1, only outgoing CPD requests to a platform's predefined detection server should be allowed. Other outgoing traffic including communication with platform services should be blocked. This could be implemented by restricting networking capabilities to a dedicated CPD process and a minimal, isolated web browser instance to complete the CP sign-in, or by setting up a firewall that blocks all outgoing traffic except to the CPD server and the CP.

Stage 2: Always-on VPN Activation After the captive network is successfully remediated, the system should trigger the always-on VPN connection establishment and grant further networking capabilities to the VPN process or subsystem. If the VPN connection can be

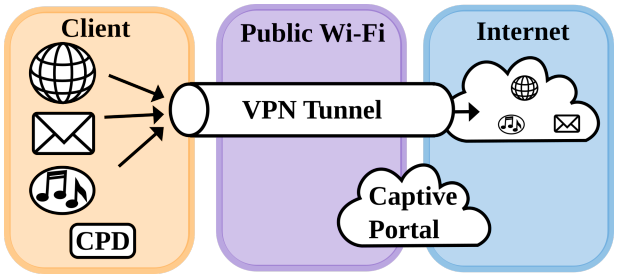


Figure 6.2. Stage 3: Only traffic to and from the VPN provider is allowed

established successfully, the bootstrapping can continue with the next stage. Otherwise, all outbound traffic should remain blocked to avoid leakage and the user should be notified.

Stage 3: Open Connectivity After a VPN tunnel is established, the system can grant networking capabilities to all other applications and services, as depicted in Fig. 6.2.

6.8 Conclusion

We are convinced that public Wi-Fis will continue to play an important role in providing mobile Internet connectivity in the future, whilst 5G and fast cellular networks become more ubiquitously available and data plans more affordable. It is therefore important that OS vendors and VPN service providers equip their software to mitigate privacy risks caused by public Wi-Fi usage. Our analyses shows that the vast majority of clients are currently unable to provide VPN establishment without leaking traffic or causing deadlocks when confronted with CPs. We propose a concept that ensures a leak-free VPN establishment in three stages and recommend OS vendors to adopt the proposal.

References

[Ali+19] Suzan Ali, Tousif Osman, Mohammad Mannan and Amr Youssef. *"On Privacy Risks of Public WiFi Captive Portals"*. In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Vol. 11737. Lecture Notes in Computer Science. Springer International Publishing, 2019, pp. 80–98. ISBN: 978-3-030-31500-9. DOI: 10.1007/978-3-030-31500-9_6.

[And] Android Developers. *"VPN"*. URL: https://developer.android.com/guide/topics/connectivity/vpn.

[Appb] Apple. *"Packet Tunnel Provider"*. URL: https://developer.apple.com/documentation/networkextension/packet_tunnel_provider.

[Appc] Apple. *"Personal VPN"*. URL: https://developer.apple.com/documentation/networkextension/personal_vpn.

[Appd] Apple. *"VPN On Demand Rules"*. URL: https : / / developer . apple . com / documentation/networkextension/personal_vpn/vpn_on_demand_rules.

[Che+13] Ningning Cheng, Xinlei Oscar Wang, Wei Cheng, Prasant Mohapatra and Aruna Seneviratne. *"Characterizing Privacy Leakage of Public WiFi Networks for Users on Travel"*. In: *2013 Proceedings IEEE INFOCOM*. IEEE INFOCOM 2013 - IEEE Conference on Computer Communications. IEEE, Apr. 2013, pp. 2769–2777. ISBN: 978-1-4673-5946-7.

[Ikr+16] Muhammad Ikram, Narseo Vallina-Rodriguez, Suranga Seneviratne, Mohamed Ali Kaafar and Vern Paxson. *"An Analysis of the Privacy and Security Risks of Android VPN Permission-Enabled Apps"*. In: *Proceedings of the 2016 ACM on Internet Measurement Conference - IMC '16*. The 2016 ACM. ACM Press, 2016, pp. 349–364. ISBN: 978-1-4503-4526-2. DOI: 10.1145/2987443.2987471.

[Kar17] Rickard Karlsson. *"EzMole: A new prototype for securing public Wi-Fi connections"*. MA thesis. Luleå University of Technology, Computer Science, 2017, p. 58.

[Kha+18] Mohammad Taha Khan, Joe DeBlasio, Geoffrey M. Voelker, Alex C. Snoeren, Chris Kanich and Narseo Vallina-Rodriguez. *"An Empirical Analysis of the Commercial VPN Ecosystem"*. In: *Proceedings of the Internet Measurement Conference 2018*. IMC '18: Internet Measurement Conference. ACM, 31st Oct. 2018, pp. 443–456. ISBN: 978-1-4503-5619-0. DOI: 10.1145/3278532.3278570.

[Kum+15] Warren "Ace" Kumari, Ólafur Guðmundsson, P Ebersman and Steve Sheng. *"Captive-Portal Identification Using DHCP or Router Advertisements (RAs)"*. RFC 7710. Dec. 2015. DOI: 10.17487/RFC7710.

[Per+15] Vasile C. Perta, Marco V. Barbera, Gareth Tyson, Hamed Haddadi and Alessandro Mei. *"A Glance through the VPN Looking Glass: IPv6 Leakage and DNS Hijacking in Commercial VPN clients"*. In: *Proceedings on Privacy Enhancing Technologies* 2015.1 (1st Apr. 2015), pp. 77–91. ISSN: 2299-0984. DOI: 10.1515/popets-2015-0006.

[Pro20] Proton Team. *"VPN Bypass Vulnerability in Apple iOS"*. 25th Mar. 2020. URL: https://protonvpn.com/blog/apple-ios-vulnerability-disclosure/.

[Som+19] Nissy Sombatruang, Lucky Onwuzurike, M. Angela Sasse and Michelle Baddeley. *"Factors influencing users to use unsecured wi-fi networks: evidence in the wild"*. In: *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*. WiSec '19: 12th ACM Conference on Security and Privacy in Wireless and Mobile Networks. ACM, 15th May 2019, pp. 203–213. ISBN: 978-1-4503-6726-4. DOI: 10.1145/3317549.3323412.

[Wik20] Wikipedia. *"Captive Portal"*. 13th Mar. 2020. URL: https://en.wikipedia.org/wiki/Captive_portal.

[WMB20] Jack Wilson, David McLuskie and Ethan Bayne. *"Investigation into the Security and Privacy of iOS VPN Applications"*. In: *Proceedings of the 15th International Conference on Availability, Reliability and Security*. ARES '20. Association for Computing Machinery, 2020. ISBN: 978-1-4503-8833-7. DOI: 10.1145/3407023.3407029.

IV

Transparency and Compliance

7 Compiling Personal Data and Subject Categories from App Data Models

Citation	Christian Burkert, Maximilian Blochberger and Hannes Federrath. *"Compiling Personal Data and Subject Categories from App Data Models"*. In: *ICT Systems Security and Privacy Protection*. Ed. by Audun Jøsang, Lynn Futcher and Janne Hagen. Vol. 625. IFIP Advances in Information and Communication Technology. Cham: Springer International Publishing, 2021, pp. 242–255. ISBN: 978-3-030-78120-0. DOI: 10.1007/978-3-030-78120-0_16
Ranking	CORE 2021 B ERA 2010 B QUALIS 2016 B1
Status	Published
Publication Type	Research Paper
Aim	The aim of this paper was to assist the complex task of creating and maintaining data protection compliance documentation about software-based processing activities (e. g., Article 30 GDPR). A major challenge is the identification and classification of personal data within the totality of all data processed by a software. To assist this task, the paper aims to provide and evaluate a semi-automatic expert process that takes advantage of entity relationship information encoded in an app's data model. By using the data model as the source of truth, the paper also aims to determine identifiability based on the fundamental software design rather than appearance or impression.

Continued on next page

Methodology	The proposed identification and classification process comprises graph-based algorithms to iteratively determine the types and paths of identifiability for all data model entities. We analytically evaluate the computational complexity and interaction cost of our proposed heuristics. We also conduct lab experiments with data models extracted from popular apps to provide empirical evidence about the practicality of our proposal.
Contribution	The paper describes and raise awareness about the problem of ubiquitous identifiability which results from the high densities in data model relationship graphs, which lead to the effect that an overwhelmingly large ratio of data model entities is related to a subject (user) and thus formally qualifies for consideration as personal data. We propose a semi-automatic expert process that identifies personal data using a novel reachability heuristic which considers the direction of relation to determine the degree of identifiability. Moreover, through expert annotations regarding the semantic of user relations (i. e., user role), we propose a way to automatically assign the identified personal data to data subject categories. Finally, we provide an evaluation of the proposal by analysing the complexity of the proposed heuristics, and demonstrate the practical applicability on real-world data models.
Co-authors' contribution	Maximilian Blochberger assisted with the design and implementation of the data model graph heuristics as well as the paper write-up. Hannes Federrath provides editorial guidance.

Abstract

Maintaining documentation about personal data processing is mandated by GDPR. When it comes to application software and its operation, this obligation can become challenging. Operators often do not know enough about app internals to be comprehensive in their documentation or follow changes enough to be up-to-date. We therefore propose a semi-automatic process to compile documentation from the source of truth: the app data model. Our approach uses data model entity relations to determine identifiability of data subjects. We guide app experts to add the semantic knowledge that is necessary to determine subject categories and to subsequently compile a condensed listing of personal data. We provide evidence for the real-world applicability of our proposal by evaluating the data models of five common web apps.

7.1 Introduction

GDPR requires operators of application software to create and maintain documentation about their processing of Personal Data (PD)[1], as stated in Article 30. Documenting such processing activities includes 'a description of the categories of data subjects and of the categories of personal data' (Article 30 (1)(e)). This requires the compilation of categorical descriptions for PD and semantic subject roles that are sufficiently detailed to, amongst others, allow regulatory authorities to assess the proportionality of data processing. However, to keep such documentation complete, correct, and up-to-date, requires great care and ongoing observation of changes to the software. Without automation, this will likely result in errors or deviation from the app as the source of truth. We therefore propose *Schemalyser*, a semi-automatic process to derive PD and subject categories from app data models. The underlying data models in the form of entities, attributes of entities and relations between entities can be extracted from various sources like source code [GBW16] or database (DB) schemes [And94] created by the app. Thereby, we are not reliant on the usage of relational DB. Schemalyser assumes that data subjects, i. e., the identifiable natural persons, are themselves represented as one or more entities in the data model, which is usually the case for (collaborative) web apps and many other client-server software. To determine which data is personal and thus needs to be included in the compliance documentation, we can therefore utilise entity relations and the information whether data is connected to a data subject entity and how. This is in accordance with Article 4(1) GDPR, which defines that, what makes personal data personal, i. e., relatable to a natural person, is less a property of the data itself but more of its semantic context and the identifiability within.

By analysing five real-world app data models, we observed a high degree of interconnectivity among entities: Entities representing data subject are directly connected with 40 to 70% of all entities and over 80% are indirectly reachable. In other words, data subjects are identifiable for almost all application data. As a result, the documentation for PD of an app would comprise

[1]We do not use the abbreviation PII to avoid confusion with non-GDPR definitions.

hundreds or thousands of attributes, thus losing its informative purpose. Moreover, a user's name is indistinguishable in terms of identifiability from the number of likes the user has received for a chat posting, which raises the question of prioritized presentation. To the best of our knowledge, we are the first to tackle this *ubiquitous identifiability* problem by introducing differentiated identifiability classes and provide a process to compile condensed PD listings per subject category.

Our main contributions are: *a*) we describe the problem of ubiquitous identifiability in data models, *b*) we propose a semi-automatic expert process to derive PD and subject categories directly from the source of truth, and *c*) we provide evidence for the complexity of real-world data models and practical applicability.

The remainder of this paper is structured as follows: We first present related work in Sect. 7.2 before we introduce the Schemalyser approach in Sect. 7.3. Section 7.4 evaluates our proposal. Finally, we discuss the integration in development workflows in Sect. 7.5 and conclude in Sect. 7.6.

7.2 Related Work

As far as we are aware, we are the first to propose a process to semi-automatically compile Article 30-related compliance documentation from app data models. Martin and Kung [MK18a] envision a data model-driven process to inventory personal data but do not name any existing approaches. Fakas et al. [FCC11] propose a similar mechanism but to semi-automatically create responses to subject access requests, whereby data entries are identified by a human keyword search. Then, starting from matching instance, the operator iteratively browses neighbouring relations, and selects entities and attributes for the response. In contrast to Schemalyser, they do not consider identifiability classes, relation-defined roles or compiling categories. Furthermore, Schemalyser analyses abstract data models and derives information about instances that can potentially exist.

A similar approach is used in [HTM19] to generate Record of Processing Activitiess (RPAs) from formal Enterprise Architecture models and derive data recipients based on relations between business processes. However, individual business processes and their categories of PD and subjects are documented through expert interviews and out of scope in terms of composition. Regarding RPA best practices especially concerning the categories of PD and subjects, a recent analysis of RPA templates in [RPB20] did not include categorisation aspects into their semantic model. Bercic and George [BG09] discuss identifiability in DBs from a legal pre-GDPR view and also conclude that all data that is linkable to a subject within a DB should be considered PD.

7.3 Schemalyser Approach

Our approach builds on a graph representation of the data model where the vertexes represent the data model entities and the edges represent directed 1:N relations between entities, i. e., foreign key (FK) relations. Since there can be multiple FK relations between the same two entities, the result is a multigraph. We call this graph a *scheme*: A scheme S is defined as a multigraph $S = (E, R)$ with a set of entities E and FK relations $R \subseteq E \times E \times \mathbb{N}$, where $(e_1, e_2, i) \in R$ denotes the i-th FK from entity e_1 to entity e_2. Note that entity attributes are left out for simplicity at this stage. For legibility, we will use the notation $(e_1, e_2) \in R$ to express that there exists an $i \in \mathbb{N}$ such that $(e_1, e_2, i) \in R$.

Based on this graph representation, our approach takes four partially automated steps to derive a condensed listing of PD per subject category, which are described in detail in the following.

7.3.1 Seed Identification

To determine what data within the scheme is potential PD, we first need to identify one or more *seed* entities: A seed $s \in E$ is an entity in the data model that represents a data subject or a role (e. g., user, reporter) and is modelled as a dedicated entity. A scheme can contain multiple seeds. Correctly and completely identifying all seeds requires domain knowledge about the app data model. However, we find that seeds typically stand out as central entities within the data model (cf. Sect. 7.4.1). Thus, to speed up their identification, we propose to rank the entities according to their degree centrality.

7.3.2 Identifiability Markup

Given the seeds as representations of data subjects in the data model, the second step now determines for the remaining entities a notion of identifiability with respect to each seed. In other words: does the data model associate a given entity with one or more of the seeds and if so how.

The naive approach would be to ignore the direction of relations and calculate the reachability graph of each seed s in an undirected scheme. We call this the partition $P_s \subseteq E$, which is defined recursively from base $\{s\}$ as:

$$P_s = \{s\} \cup \{e_2 \in E \mid \exists e_1 \in P_s \colon (e_1, e_2) \in R \vee (e_2, e_1) \in R\}$$

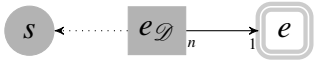
However, this simplistic approach neglects the cardinalities implied by the directionality of the relations, i. e., the direction implies whether an entity is related to exactly one instance of another entity or an arbitrary number of such instances. This distinction determines whether a relation to a seed allows to single out an individual data subject or a potential multitude of data subjects. Following this distinction, we subsequently formalise the direction semantic and introduce classes of identifiability.

Relation Direction Semantics

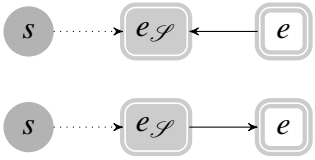
We distinguish two semantic sub-relations between a pair of entities depending on the direction of the FK relation: *extending* and *providing*:

1. *Extending* $e_1 \rightarrow e_2$ (n-to-1): Each instance of e_1 extends the context of exactly one instance of e_2 and is unambiguously associated with that instance.
2. *Providing* $e_1 \leftarrow e_2$ (1-to-n): An instance of e_1 can provide non-exclusive context to an arbitrary number of instances of e_2.

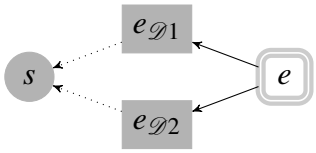
(a) Dedicated: n instances of entity e are related to a single instance of $e_{\mathscr{D}} \in \mathscr{D}_s$.



(b) Shared 'Original': An instance of e provides context for n instances of $e_{\mathscr{D}} \in \mathscr{D}_s$.



(c) Shared 'Extended': Entity e further extends or provides to shared entity $e_{\mathscr{S}} \in \mathscr{S}_s$.



(d) Shared 'Merged': Entity e is shared by extending s via multiple $e_{\mathscr{D}1}, e_{\mathscr{D}2}, \ldots \in \mathscr{D}_s$.

Figure 7.1. Identifiability classes for a seed from the perspective of other entities.

Identifiability Classes

Based on these relation semantics, we identified three class of identifiability of a seed s regarding another entity. The classes are not necessarily exclusive and are recursively defined as subsets of E as follows:

1. *Dedicated*: An entity is dedicated to a seed if it has a direct, extending FK path to the seed (see Fig. 7.1a):

$$\mathscr{D}_s = \{s\} \cup \{e \in E \mid \exists e_{\mathscr{D}} \in \mathscr{D}_s \colon (e, e_{\mathscr{D}}) \in R\}$$

   Example: A user (seed) has multiple addresses, e. g., invoice and delivery.

2. *Shared*: An entity has one or more indirect paths to the seed by providing for a dedicated entity (*original share*), by extending or providing for another shared entity (*extended share*), or by extending dedicated entities through two or more distinct FK paths (*merged*

share), see also Figs. 7.1b to 7.1d:

$$\begin{aligned}
\mathscr{S}_s &= \mathscr{S}_s^{Orig} \cup \mathscr{S}_s^{Ext} \cup \mathscr{S}_s^{Merged} \\
\mathscr{S}_s^{Orig} &= \{e \in E \mid \exists e_{\mathscr{D}} \in \mathscr{D}_s \colon (e_{\mathscr{D}}, e) \in R \\
&\qquad \wedge (e_{\mathscr{D}} = s \vee \exists e_{\mathscr{D}'} \in \mathscr{D}_s, e_{\mathscr{D}'} \neq e \colon (e_{\mathscr{D}}, e_{\mathscr{D}'}) \in R)\} \\
\mathscr{S}_s^{Ext} &= \{e \in E \mid \exists e_{\mathscr{S}} \in \mathscr{S}_s \colon ((e, e_{\mathscr{S}}) \in R \vee (e_{\mathscr{S}}, e) \in R) \\
&\qquad \wedge (\exists e_{\mathscr{D}} \in \mathscr{D}_s, e_{\mathscr{D}} \neq e \colon (e_{\mathscr{D}}, e_{\mathscr{S}}) \in R \\
&\qquad \vee \exists e_{\mathscr{S}'} \in \mathscr{S}_s, e_{\mathscr{S}'} \neq e \colon ((e_{\mathscr{S}'}, e_{\mathscr{S}}) \in R \vee (e_{\mathscr{S}}, e_{\mathscr{S}'}) \in R))\} \\
\mathscr{S}_s^{Merged} &= \{e \in E \mid \exists e_{\mathscr{D}1}, e_{\mathscr{D}2} \in \mathscr{D}_s \colon \{(e, e_{\mathscr{D}1}), (e, e_{\mathscr{D}2})\} \subseteq R\}
\end{aligned}$$

In $\mathscr{S}_s^{Orig}$ and $\mathscr{S}_s^{Ext}$ constraints are necessary to avoid that a *shared* classification propagates backwards to entities that caused it. In Fig. 7.1b, for instance, $e_{\mathscr{D}}$ does not receive the extended shared membership from e, because $e_{\mathscr{D}}$ is the sole origin of e's shared membership. Also note that the merged share classification is assigned after all entities have been otherwise classified and the dedicated, shared original and shared extended sets are stable. It does not cause further extended shares.

Example: Multiple users are participants in a chat room. Information about the chat room is attributable to all participants without being able to single out one participant.

3. *Unrelated*: There is no relation between the seed and the other entity:

$$\mathscr{U}_s = E \setminus P_s$$

Example: Global configuration of the application.

Based on these three identifiability classes, we can distinguish whether an attribute, according to the entity it belongs to, solely concerns an individual data subject (dedicated), potentially relates to a set of data subjects (shared), or can most likely be excluded from consideration as PD (unrelated).

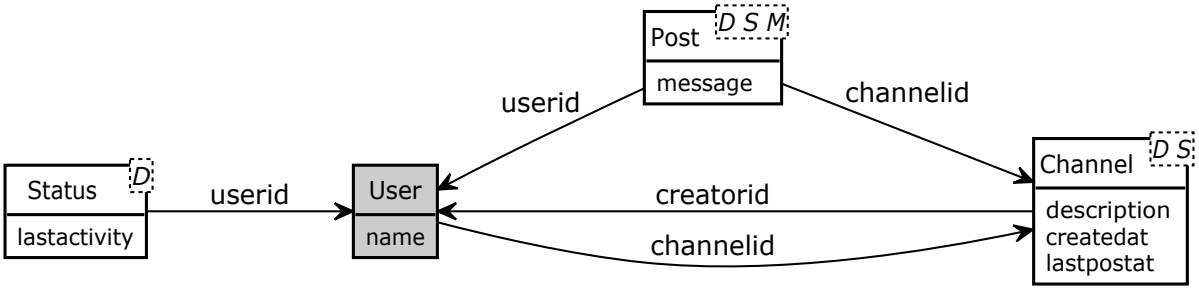


Figure 7.2. Simple chat app data model with identifiability classes (D)edicated, (S)hared, and shared (M)erged assigned to each data model entity.

Example

To illustrate the classification, Fig. 7.2 shows the resulting identifiability classes for a data model of a simple chat app. In this example, the User entity is the seed. The other entities all have at least one dedicated path to User and therefore receive the `D` label. The relation between User and Channel through channelid is providing, hence Channel receives the original shared label `S`, which is then carried over as extending shared to Post. And finally, as Post has two dedicated paths, one direct and one via Channel, it receives the merged shared label `M` which implies `S`, because each dedicated path might associate a different user to Post that way, all of which then share Post's attributes.

7.3.3 Role Determination

During this step, a domain expert with knowledge about the data model is guided to determine roles of a user that are reflected in the data model. These roles eventually build the basis for categorising data subjects and grouping PD. We regard a role as a semantic distinction of a user's relation to the system that is characterised by the user engaging with the system in an optional manner. By being optional, it differentiates itself from a general engagement with the system, i. e., the default user role. For instance, a user chooses to report an issue by which they take the role of an issue reporter. In the data model, roles can be modelled in different ways, e. g., by having dedicated role entities or explicit FK attributes for a role. We find that roles are typically defined by the entities and attributes directly surrounding the seed, i. e., the first hops on a path between the seed and other entities. We call those *first-hop relations*. However, not every first-hop relation necessarily constitutes a role, because it might lack a defining characteristic like the optionality of the engagement. This is a semantic distinction that cannot generally be inferred solely based on the scheme. Therefore, during this step of determining roles, a domain expert has to classify, which first-hop relations define a role. For this classification process, we propose the following classes:

- *Integral*: The relation adds data for users in general that is integral and unconditional to the overall system functionality.
- *Role*: The relation describes an optional user engagement with the system that defines their capabilities and perception by the system and other users.
- *Conditional*: The relation provides details related to the usage of an optional feature which, other than a role, does not define their perception because it is less functionally significant or visible.

The distinction between roles and conditionals is gradual. Relations like the uploader of a file attachment to a post could be regarded as both an uploader role or a conditional usage of the feature to attach files to a post. We argue that, in this example, uploader should be considered a conditional because it is less significant and visible than the poster role which coincides with uploading a file. As roles will later form subject categories that should be meaningful to non-experts, they should be defined sparingly.

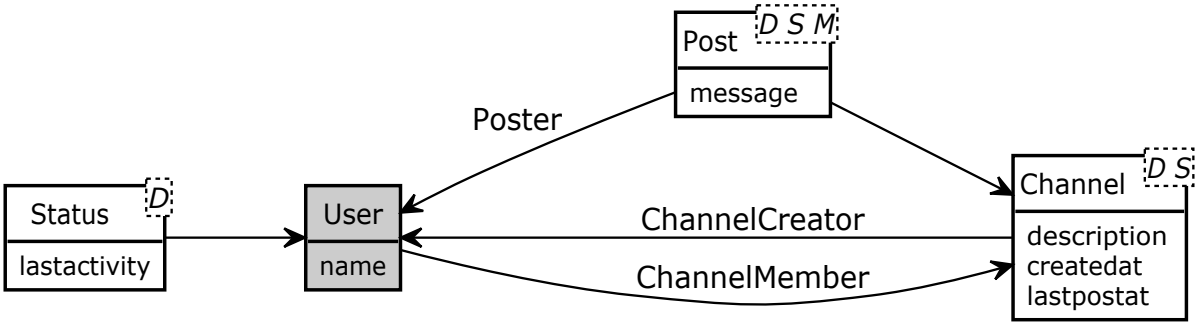


Figure 7.3. Minimal chat app data model with three determined roles.

Revisiting the chat app example in Fig. 7.3, we deemed three out of the four first-hop relations as role-defining. The fourth relation between Status and User is obligatory for each user and thus integral.

7.3.4 Decisive Role Selection

During this step, a domain expert selects for each attribute of every entity with a dedicated and/or shared classification, which of the previously determined roles are decisive for this attribute. We consider a role as *decisive* for an attribute if the attribute's value is derived from a property or action of a member of that role. Potential candidates for decisive roles are all roles whose defining relation lies on a path from the respective entity to the seed. Note, that as we are only considering simple paths, i. e., paths where each node is at most visited once, there is always only one first-hop relation and thus only one role-defining relation on each path. We call the candidates that are deemed as non-decisive *subjected roles*. Subjected roles might be equally identifiable than the decisive role of an attribute. The distinction rather allows to prioritise and filter attributes as will be shown in the final step. For instance, a channel creator is subjected by messages of posts in their channel but does not decide them. In the following, we describe the substeps of enumerating paths and selecting candidates as well as lower-complexity alternatives for large data models.

Path Enumeration

To determine the candidates for decisive roles for a given seed and entity, we first have to find all paths between that seed and entity. However, we cannot simply list all simple paths between these nodes, because the paths have to account for the relation direction semantics described in Sect. 7.3.2 and the resulting propagation of identifiability classes. We denote the paths through which entities received their dedicated or shared classification as dedicated and shared paths, respectively.

As the propagation of the dedicated class follows only extending edges that are directed towards the seed, dedicated paths are strictly directed walks in the graph. Hence, dedicated paths are

Entity	$\mathscr{D}$ Paths	$\mathscr{S}$ Paths
Status	$\{U \leftarrow S\}$	$\{\}$
Post	$\{U \leftarrow P, U \leftarrow C \leftarrow P\}$	$\{U \rightarrow C \leftarrow P\}$
Channel	$\{U \leftarrow C\}$	$\{U \rightarrow C, U \leftarrow P \rightarrow C\}$

Table 7.2. Enumeration of dedicated and shared paths for the chat app example.

simple paths in a directed scheme graph with inverted directions. Regarding shared paths, we have to consider graph walks with mixed directions. However, as defined in Sect. 7.3.2, a shared path contains at least one providing relation and allows no back-propagation. Shared paths are therefore a subset of paths in an undirected graph. To illustrate this, Table 7.2 enumerates the dedicated and shared paths of the chat app from Fig. 7.4. Note that in this example, ignoring the constraints for shared paths would for instance incorrectly add the shared paths $U \leftarrow P$ and $U \leftarrow C \leftarrow P$ for entity Post, which would lead to an enlarged and misleading PD listing for the roles Poster and Channel Creator in the final step.

Candidate Selection

Candidates for a given entity are all roles whose defining relation lies on of the previously enumerated dedicated or shared paths. For large data models, the number of candidates might still be quite high. To speed up the manual selection process by the domain expert, we propose to present the candidates in descending order according to the following selection likelihood heuristic:

1. dedicated-only roles before mixed before shared-only roles, then
2. roles with shortest paths first, then
3. roles with lowest number of paths first.

We argue that dedicated-only roles are more likely decisive than mixed and shared-only roles because an identifiable individual is more likely the originator of an action or actively involved than a set of individuals. Regarding the influence of path length, we argue that shorter paths imply a more direct logical connection. The third ordering rule presumes that a higher number of paths using the same first-hop relation decreases the semantic specificity and is thus more likely to be a modelling artefact than semantically significant.

In completing this step, the domain expert selects one or more decisive roles for each attribute. Figure 7.4 shows the decisive role selection for the attributes of the chat app example. The Poster role is selected as decisive for the message of a post. The attributes of Channel have each a different decisive role. This selection is modelled after the Mattermost chat app, where the ability to set and change the description of a channel is given to every member of that channel.

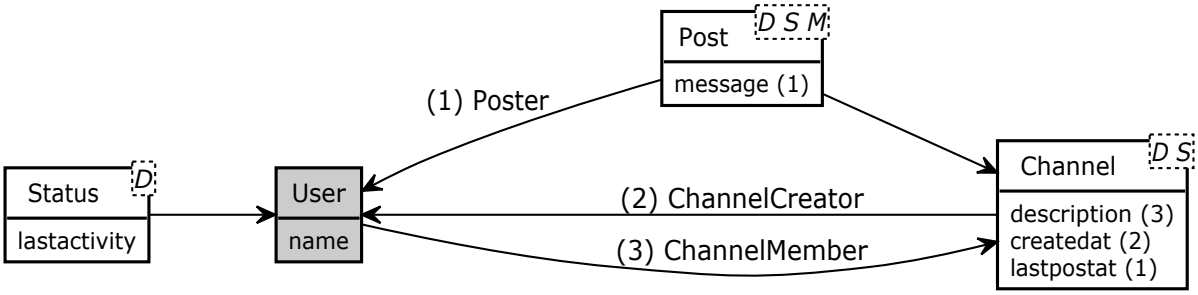


Figure 7.4. Minimal chat app data model with decisive roles selected.

Complexity and Path Enumeration Alternative

Listing all paths between the seed and the other entities has a $O(|E|!)$ computational and spacial complexity. For large and highly interconnected data models, it might therefore be practically infeasible to use the path enumeration approach, as we will show in Sect. 7.4. For those cases, it might be a sufficient compromise to determine candidates by checking for reachability via each role-defining relation instead. To do so, we check for each role-defining relation which entities are still reachable from the seed if all but this first-hop relation were removed. Given that the complexity of a reachability check between a single pair of entities is $O(|E|+|R|)$, the complexity for checking the reachability of a seed and all other entities via each role-defining relation is therefore $O(r|E|(|E|+|R|))$, where r is the number of role-defining relations. In the worst case of a fully meshed graph where $r = |E|$ and $O(|R|) = O(|E|^2)$ the complexity becomes quadratic. Despite reachability checking being a more efficient alternatives for large inputs, we recommend the path enumeration where possible, because being able to inspect a path can provide valuable context information to the domain expert.

7.3.5 Condensed PD Listing

For this final step, we propose a condensed per-role listing of PD to maintain readability and focus on those attributes that have a higher significance for a given role. To achieve this condensed listing, we use the dedicated/shared distinction and the decisive role selections to determine if an attribute should be listed as PD for a given role in detail or if it can be aggregated into a *grouping term*. To list PD for a given role, we list an attribute explicitly if the role is decisive for that attribute. If not, the attribute is represented by a grouping term. As a result, we have four sets of PD for each role that decrease in identifiability and significance:

1. *Dedicated decisive*: The attribute is significantly determined by a member of this role and the member can be singled out.
2. *Dedicated non-decisive*: The attribute is associated with an identifiable member of this role without being determined by them.
3. *Shared decisive*: The attribute is significantly determined by a member of this role but *no* member can be singled out.

4. *Shared non-decisive*: The attribute is not determined by this role and *no* member can be singled out.

The grouping terms used in the non-decisive sets can represent a subset of attributes of an entity, an entity as a whole, or multiple semantically related entities. A display name of an entity, i. e., a name dedicated for presentation to users, could for instance be used as an automatically derived term. In cases where such derived grouping terms might be too incomprehensible for the general public, a domain expert should assign fitting grouping terms manually. If the media supports hypertext, grouping terms should be expandable to the individual attributes summarised by them.

To illustrate the condensed listing, we again use the data model and roles from the chat app as shown in Fig. 7.4. The resulting PD listing is shown in Table 7.3. Entity names are used as grouping terms.

Role	1. $\mathscr{D}$ decisive	2. $\mathscr{D}$ non-dec	3. $\mathscr{S}$ decisive	4. $\mathscr{S}$ non-dec
User	status lastactivity	-	-	-
Poster	message lastactivity	-	-	*Channel data*
Channel Creator	createdat	*Channel data* *Post data*	-	-
Channel Member	-	-	description	*Channel data* *Post data*

Table 7.3. Condensed PD listing for the chat app. Cursive entries are group terms.

7.4 Evaluation

In this section, we evaluate the practicality of our proposal in terms of number and difficulty of necessary interaction, the complexity, as well as the degree of condensation achieved in the PD listing. To also gain an insight into costs for real-world apps, we used five common web apps (see Table 7.5) and analysed their data models. Lacking the necessary domain expertise for the software architecture of all those apps, we were only able to evaluate the manual steps for the Mattermost app, which we have extensively studied before [BF19].

7.4.1 Interaction Cost and Complexity

In this section, we assess the number of manual actions and the number of alternatives a domain expert has to choose from during each process step. An overview of the costs and complexities is provided in Table 7.4.

Step	#Actions	#Decisions	Worst Case Complexity
1. Seed Identification	1	$O(1)$	$O(\|E\|^2)$
2. Identifiability Markup	-	-	$O(\|E\|^2(\|E\|+\|R\|)\deg_{\max})$
3. Role Determination	$O(\|E\|)$	3	-
4. Decisive Role Sel.	$O(\|E\|)$	$O(\|E\|)$	$O(\|E\|!)/O(\|E\|^4)$
5. PD Listing	$O(\|E\|)$	1	$O(\|E\|^2)$

Table 7.4. Overview of the cost and complexity of our proposal.

Seed Identification

This requires one manual step to select all seeds from a list of all entities. However, we find that seeds typically exhibit a high degree centrality and rank in the top two entities with the highest degree centrality in the tested apps, ranging from 41 to 69%. Therefore, the expert does not need to inspect all entities at this point. Calculating the degree centrality over the graph adds a $O(|E|^2)$ complexity. Note that in the following, we will assume that the number of seeds in a data model does not grow with the number of entities. Instead, we argue that the number of seeds depends on modelling styles and is typically in the low single digits. Our evaluation examples support this assumption having each only a single seed.

Identifiability Markup

Assigning identifiability classes is a fully automatic process. Our proof-of-concept implementation uses a breadth-first (BFS) approach to propagate the classes through the scheme graph. It repeats until the classes are stable, i. e., until it has gathered for each entity and identifiability class, all neighbouring entities that propagate that label to that entity. Note, that gathering all propagation origins is necessary to avoid back propagation (cf. Sect. 7.3.2). During a single BFS traversal we process each entity's neighbours, which leads to a complexity of $O((|E|+|R|)\deg_{\max})$ complexity, where $\deg_{\max}$ is the maximum node degree. To reach stability, a theoretical worst case of $|E|^2$ repeats would be necessary if every entity receives its classes from every other entity but with only a single new propagation per iteration. However, this is a very conservative estimation, since an increase in connectivity would also speed up propagation. In practice, reaching stability took 4 to 5 iterations for our test apps. In total, this step has at worst a $O(|E|^2(|E|+|R|)\deg_{\max})$ complexity.

Role Determination

During this step, all first-hop relations of every seed have to be inspected and categorised. In the worst case, if all seeds have disjoint first-hop relations and are fully connected, this takes $n_s|E|$ steps, where n_s is the number of seeds. During each step, one of three classes has to be selected. As an insight into real-world first-hop relation counts, Table 7.5 lists the degree

Application	Full Scheme			Partition P_s				Ident. Cls.			Paths	
	$\|E\|$	$\|R\|$	D[%]	$\|P_s\|$	$\|R_s\|$	D[%]	$\deg(s)$	$\|\mathscr{D}\|$	$\|\mathscr{S}\|$	$\|\mathscr{D}\cap\mathscr{S}\|$	Ded.	Shared
Bugzilla 5.0	76	102	1.79	61	100	2.73	35	46	47	32	175	105908
Gitlab 12.7.5 CE	308	671	0.71	289	669	0.80	128	269	282	262	2997	>1M
Mattermost 5.18	40	54	3.46	32	54	5.44	27	28	23	19	41	1439
Taiga 5.5.7	68	116	2.55	61	114	3.11	29	54	56	49	238	103943
Zulip 3.2	77	116	1.98	66	116	2.70	46	45	54	33	54	226690

Table 7.5. Evaluation of automatic steps for sample data models.

$\deg(s)$ for each app's seed. We find that, as discussed in the seed identification step before, that seeds are typically highly connected with degrees roughly around half of $|E|$.

Decisive Role Assignment

Regarding interaction cast, the domain expert selects decisive roles for every attribute of every entity in a seed partition. For each attribute, all roles that lie on a path to that entity need to be considered. In the worst case of a fully connected graph, this requires $O(|E|)$ steps and $O(|E|)$ decisions each. Computationally, the selection of decisive role candidates requires either a path enumeration or reachability checks, which, as discussed at the end of Sect. 7.3.4, result in factorial or quadratic cost, respectively.

The selection of decisive roles can be partly automated if an entity has only a single role candidate. Integral-only cases can be auto-assigned to a generic user role, as can be conditional-only cases with additional info about the condition. Otherwise the domain expert has to manually select from the list of candidates. Table 7.6 provides the number of single-candidate and integral-only entities as well as the average number of candidates available for the non-automatable entities.

Application	$\|P_s\|$	$\|R_s\|$	$\deg(s)$	Roles	Integral	Single Cand.	Avg. Cand.
Chat App Example	4	5	4	3	1	0	3
Mattermost	32	54	27	13	10	5	9

Table 7.6. Decisive role selection cost indicators for two practical examples.

PD Listing

If grouping terms cannot be automatically derived from already available display names or descriptions, an expert has to manually assign $|E|$ terms if entities are not aggregated. Besides the grouping term assignment, the list construction is automatic. The construction inspects every entity and adds their attributes or grouping terms to the four sets of every role. With a worst case of $|E|$ roles this leads to a $O(|E|^2)$ complexity.

7.4.2 Degree of Condensation

The degree of condensation depends of course on the composition of the data model. Our approach condenses attributes for which a role is not decisive. Consequently, the degree of condensation is inversely proportionate to the ratio of decisive roles to role candidates. Hence, if every role is decisive for every connected attribute, the condensation would be minimal. Applying this worst case to the chat example from Table 7.3, the attributes of Channel and Post would be listed for each of the three roles resulting into 14 attributes and zero grouping terms instead of the original 6 and 5. The condensation by the grouping terms depends on their defined scope, in this example an entity. Larger grouping scopes naturally result into fewer grouping terms per subject category.

7.5 Integration into Development Workflows

We argue that the best way to keep PD compliance documentation up-to-date and in sync with the app as its source of truth is to integrate Schemalyser into the development workflow. Thereby, vendors could offer PD listings like in Table 7.3 in a machine-readable form as templates for customers' compliance documentation. In the following, we describe, how, by using code annotations, necessary domain knowledge could be added to further increase automation, and how Schemalyser can be used to monitoring changes to the data model.

Annotation

Our approach relies on expert knowledge about the app architecture, mainly to determine and select decisive roles. Ideally, this information is noted by experts in a machine-readable way, e. g., in the form of code annotations, such that tools like ours can utilise it. Listing 7.1 shows how such an annotation might look like for the Channel class of the running example in Python. The `roledef` annotation defines a (FK) attribute as a new role. The `role` annotation assigns such a defined role as decisive role for the given attribute.

Listing 7.1. Exemplary code annotation for roles and assigned decisive roles.

```
class Channel:
    creator_id: int     # roledef: ChannelCreator, default
    created_at: int
    description: str    # role: ChannelMember
    lastpost_at: int    # role: Poster
```

Compliance Monitoring

As a way to review or monitor compliance during development, Schemalyser could be integrated into testing: While tests are setting up a DB, a scheme could be dumped, pushed to the Schemalyser service, where it is compared to dumps of previous test runs and changes are flagged for review to ensure that additions of PD or identifying relations are intentional and compliant. Such change detection reapplies previous classifications to the new scheme whereupon new first-hop relations or role candidates trigger a review.

7.6 Conclusion

We have proposed a novel semi-automatic process to compile PD and subject categories, and condensed PD listings on the basis of app data models and entity relations. By analysing real-world app data models, we have pointed out the need for this condensed listing of PD to counter the effects of ubiquitous identifiability in data models, where over 80% of entities are attributable to data subjects. We argue that correctly assigning PD to subject categories requires a degree of architectural knowledge that is likely exclusive to the software vendor. We encourage vendors to add annotations about subject roles to their source code and follow our decisive role approach to make their knowledge accessible to customers in a machine-readable way and allow a further automation of compiling comprehensive and up-to-date compliance documentation.

Acknowledgements

The work is supported by the German Federal Ministry of Education and Research (BMBF) as part of the project Employee Privacy in Development and Operations (EMPRI-DEVOPS) under grant 16KIS0922K.

References

[And94] Martin Andersson. *"Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering"*. In: *Entity-Relationship Approach — ER '94 Business Modelling and Re-Engineering*. Red. by Gerhard Goos, Juris Hartmanis and Jan Leeuwen. Vol. 881. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1994, pp. 403–419. ISBN: 978-3-540-49100-2. DOI: 10.1007/3-540-58786-1_93.

[BF19] Christian Burkert and Hannes Federrath. *"Towards Minimising Timestamp Usage In Application Software: A Case Study of the Mattermost Application"*. In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Vol. 11737. Lecture Notes in Computer Science. Springer International Publishing, 2019, pp. 138–155. ISBN: 978-3-030-31500-9. DOI: 10.1007/978-3-030-31500-9_9.

[BG09] B. Bercic and C. George. *"Identifying Personal Data Using Relational Database Design Principles"*. In: *International Journal of Law and Information Technology* 17.3 (1st Sept. 2009), pp. 233–251. ISSN: 0967-0769, 1464-3693. DOI: 10.1093/ijlit/ean007.

[FCC11] Georgios John Fakas, Ben Cawley and Zhi Cai. *"Automated Generation of Personal Data Reports from Relational Databases"*. In: *Journal of Information & Knowledge Management* 10.02 (June 2011), pp. 193–208. ISSN: 0219-6492, 1793-6926. DOI: 10.1142/S0219649211002936.

[GBW16] Sandra Greiner, Thomas Buchmann and Bernhard Westfechtel. *"Bidirectional Transformations with QVT-R: A Case Study in Round-Trip Engineering UML Class Models and Java Source Code"*. In: *MODELSWARD 2016 - Proceedings of the 4rd International Conference on Model-Driven Engineering and Software Development, Rome, Italy, 19-21 February, 2016*. SciTePress, 2016, pp. 15–27. DOI: 10.5220/0005644700150027.

[HTM19] Dominik Huth, Ahmet Tanakol and Florian Matthes. *"Using Enterprise Architecture Models for Creating the Record of Processing Activities (Art. 30 GDPR)"*. In: *2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC)*. 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC). IEEE, Oct. 2019, pp. 98–104. ISBN: 978-1-72812-702-6. DOI: 10.1109/EDOC.2019.00021.

[MK18a] Yod-Samuel Martin and Antonio Kung. *"Methods and Tools for GDPR Compliance Through Privacy and Data Protection Engineering"*. In: *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). IEEE, Apr. 2018, pp. 108–111. ISBN: 978-1-5386-5445-3. DOI: 10.1109/EuroSPW.2018.00021.

[RPB20] Paul Ryan, Harshvardhan J. Pandit and Rob Brennan. *"A Common Semantic Model of the GDPR Register of Processing Activities"*. In: *Legal Knowledge and Information Systems - JURIX 2020: The Thirty-Third Annual Conference, Brno, Czech Republic, December 9-11, 2020*. Vol. 334. Frontiers in Artificial Intelligence and Applications. IOS Press, 2020, pp. 251–254. DOI: 10.3233/FAIA200876.

V

Conclusion

8 Thesis Summary

I set out in my dissertation project to understand how user activities are temporally exposed through apps, and how this can be mitigated to improve user privacy. I chose this topic because it seemed to me a pressing issue, gaining relevance with the rapid integration of digital and centralised services into our work processes. Just from using such tools and services daily as software developer and researcher, I could observe that my and my colleagues' activity patterns are exposed. The resulting privacy infringement seemed apparent: Why is it necessary that everyone can see when I did something? What is this information even used for? And yet, the topic of activity time exposure through apps had not found much attention from privacy researchers.

I noticed early on in my research, that this problem is very faceted and at the intersection of applied data protection, software engineering, and data protection law. This interdisciplinarity is reflected in the different areas of research, I was able to make contributions to with this work. My explorative work to *understand* the usage of timestamps and users' precision demands proposes a more purpose-centric focus within software design and privacy auditing. My contributions also touch the area of data mining and quantifiable privacy, where I developed metrics to demonstrate the privacy impact of minimisation. The constructive work to *mitigate* activity time exposure, was able to contribute to the area of application and system security as well as privacy by design tooling. And lastly, to *maintain* privacy compliance, I proposed methods to bridge some of the gap between legal compliance obligations and real-world software complexities.

In the following, I revisit my research questions and reflect on the answers I was able to provide in my dissertation. After that, I draw a bigger picture of the contributions of my dissertation. My thoughts on the implication of my work and the limitations that might stand in the way of a larger impact are discussed last.

8.1 Revisiting the Research Questions

At the beginning of this thesis, I presented five research questions covering my activities in various areas of activity time exposure. To conclude the thesis, this section revisits each research question and briefly recaps the answers I could find in my work.

RQ1: How are activity timestamps used in practice by apps and for what purposes?

To answer this question, we designed and executed a white box analysis of source code to classify all usage purposes of timestamps. We found that white box case studies are a rich source for analysing timestamp usage. In two case studies, we identified timestamp usage archetypes that appear frequently and clearly have the potential for data minimisation. I could observe apparent patterns of timestamp usage that suggest that developers follow privacy anti-patterns. I most prominently noticed a pattern to include state-related timestamps in many if not all data model entities even though they are unused. Our chosen methodology was, however, not suited to also investigate the motivation or mental model behind the inclusion of such timestamps.

RQ2: What is the user demand for informative activity timestamps and how can it be determined?

I described informative activity timestamps as a special type of timestamp purpose whose potential for data minimisation is difficult to assess because of the inherent subjectivity and vagueness of human information processing. I proposed to assess data minimisation potential by empirically estimating user demand for timestamp precision. By analysing user preferences for Git, I could demonstrate that empirical data suggests potential for data minimisation.

But how small a data minimisation is even worth the effort? How much does user privacy improve thorough minimisation? These obvious questions were raised by many confronted with our findings. And of course, I asked myself these questions. However, to answer the question, I needed something to measure and contrast 'privacy' before and after minimisation. I contributed an approach to assess privacy loss in chronological activity timelines by means of counting the discernible inter-activity intervals. Using this approach, I was able to analyse a large-scale GitHub commit dataset and empirically evaluate the effect of timestamp precision reduction on discernible inter-activity intervals. The results suggest that even comparatively low reductions (up to an hour) have a significant privacy impact. This finding surprised me.

RQ3: How can frameworks be designed to help developers minimise timestamp exposure?

We could translate concepts and privacy patterns from the literature and my prior work from the Mattermost analysis into a design of timestamp alternatives. We found that the most common usage archetypes can be met with a few essential alternatives featuring precision reduction and counters. Our design validation case study supported this finding, but also highlighted that additional efficiency can be gained by providing alternatives that combine multiple usage archetypes into one replacement. In providing these combined alternatives, we could regain some of the versatility of conventional timestamps while remaining more privacy friendly. We designed the APIs for our timestamp alternatives to guide developers towards making deliberate

and explicit choices. The alternatives do not offer any presets for timestamp precision, but instead force to set case-specific levels. How this effects developer behaviour has to, however, be investigated by future work.

In terms of cost, we could demonstrate that timestamp alternatives are not more costly than conventional timestamps if we are content with basic use cases like generalised dates and chronological ordering. Functionality to gradually reduce precision over time is naturally more resource intensive. Similarly, the effort to adapt our alternatives is very little for functionally equivalent alternatives. In any case, we could demonstrate that the effort is manageable even without prior knowledge of the codebase.

RQ4: What can mitigate the observability of mobile workers' activities by network-based attackers?

We could demonstrate that relying on VPNs to protect mobile workers' traffic from network-based eavesdroppers is flawed. Our experimental analyses revealed that even specialised Personal VPN apps fail in the vast majority of cases to accommodate for common hurdles like captive networks and as a result fall victim to traffic leaks or deadlocks.

Traffic leakage, we concluded, could be avoided if platforms, captive portal detection, and VPN apps would integrate more and cooperate better. If the three components would adhere to the strict multi-step process of network and VPN establishment, which we described, captive networks could be remediated, and VPN tunnels established without any danger of premature traffic egress or deadlocks. But without the platforms providing the necessary APIs or handing over full network control to a Personal VPN app, there is little that apps or users can do to avoid the risk of traffic leakage.

RQ5: How can personal data in app models be identified more automatically to monitor compliance?

We found that entity relationships are well-suited to assess the attributability of app data to natural persons. We proposed an interpretation of the direction of entity relationships and formalised how it induces different notions of attributability within the data model. We observed and described the effect of ubiquitous identifiability, which causes incomprehensible personal data listings due to the high relation density in data models.

We could demonstrate that the size and relation density of some real-world apps impede the practical applicability of graph-based approaches. However, this only affected one process step, which uses path listing to find likely data subject categories. As a mitigation, we proposed a compromise using less complex graph operations at the cost of some context information. We also suggested that using our Schemalyser approach during software quality assurance (QA) is a good way to keep track of changes to the attributability of data within the app. By using code annotations developers could upfront add all necessary expert declarations into the code, which lets Schemalyser run unattended as part of an automation.

8.2 Contribution in Big Pictures

I have described the contributions of my work before in detail within the introduction and within each paper. At this point, I want to take a step back and examine the contribution in bigger pictures and discuss how they play together.

I tried to express through the three pillars of my work that understanding, mitigating and maintaining go hand in hand when it comes to tackling activity time exposure. I was able to lay the groundwork of understanding developers' usage of timestamps in apps (Chapter 3). I pointed out excessive timestamping as a significant threat to user privacy in general and a driver of activity time exposure and performance monitoring. To minimise these risks, it is important that the data protection obligations purpose specification and limitation as well as data minimisation are upheld. These obligations clearly prohibit the collection of activity timestamps without any purpose and arguably also dictate to use the least privacy invasive alternative that still fulfils a given purpose with reasonable effort (see Chapter 2).

Minimising data and specifying purposes requires an understanding of procedural necessities that likely goes beyond a controller's awareness. I argued that especially for informative timestamps, i. e., with the purpose to inform users, determining the necessary level of timestamp information is complex. Nonetheless, from a regulatory standpoint, specifying a necessary level and minimising accordingly is obligatory. Otherwise, any activity timestamp collection could be justified by referring to the soft purpose of user information, thus thwarting accountability.

I proposed the approach to determine the necessary level of precision for informative timestamps by way of analysing how users set up their tools (Chapter 4). I demonstrated this for a large user configuration dataset, where I deduced the precision levels that users apparently deemed sufficient by choosing corresponding settings. For this dataset, my analysis revealed a vast majority of users that does not utilise high-precision timestamps. With that analysis, I contributed the first empirical evidence of user demand for timestamp precision, which can be used to inform a more appropriate selection of the default precision of informative timestamps.

To more easily allow developers to realise these minimisation potentials in their app design, we contributed a framework of alternatives to conventional timestamps (Chapter 5). It naturally comprises alternatives for reduced-precision timestamps, usable to minimise informative timestamps, but also alternatives to address those manifestations of excessive timestamping, where timestamps are used for programmatic functions that can be performed without the sensitive properties of a timestamp.

How sensitive activity timestamping is to user privacy is hard to quantify, because the privacy impact always results from higher-order conclusions that are context specific, and not from abstract timestamps. Nonetheless, having concrete privacy impact measures is useful to judge the effect of minimisation efforts. I contributed such an impact measure on the basis of observable inter-activity intervals (Chapter 4). I used this measure to empirically demonstrate the effect of timestamp precision reduction on real-world user activity data. With this analysis,

I contributed the first evidence for the expectable impact of data minimisation of activity timestamps.

Minimising activity time exposure can, however, not be confined to stored model data only. I argued that users are also vulnerable to exposing their activities whenever they use untrusted infrastructure like public Wi-Fi hotspots (Chapter 6). In experimental analyses, I could demonstrate that VPN apps as protective measures are not sufficiently equipped to reliably protect the confidentiality and availability of the communication when facing common hotspot configurations. I contributed the first such analysis of popular Personal VPN apps and reported various security flaws to the app and platform vendors. The findings show that avoiding activity time exposure on a network level still requires a better integration and cooperation between platforms and third-party apps.

Being able to assess which activity data is collected for what data subject categories requires specific and comprehensive information about the personal data processed by an app. It also requires a meaningful mapping of the personal data to the data subject categories. We contributed a process to compile such information from standard data model descriptions (Chapter 7). By analysing real-world app data models, I could, however, demonstrate that the aim of comprehensiveness is significantly complicated by the risk of being incomprehensible. This risk of incomprehensibility, I discovered, is driven by the effect of ubiquitous identifiability which originates from high relation densities in data models. I introduced novel semantic distinctions for the attributability of personal data, which offer new options to discover more relevant data categories within the totality of attributable data. Our approach provides a way to highly automate the compilation of comprehensive compliance information while also reducing the overwhelming effect of ubiquitous identifiability. This enables developers and auditors to automatically monitor for changes in the attributability of app data and in particular the introduction of newly collected activity data.

8.3 Implications and Limitations

In this section, I want to reflect on the implications that my contributions could have on how data minimisation is practiced in software design and also on privacy engineering and policy. I also want to be clear about the more general limitations that my work has and which might hinder its impact.

8.3.1 Excessive Timestamp Usage

Evidence for excessive timestamping was found in every app that we inspected. Overall, we performed two in-depth case studies and several briefer analyses of timestamp usage. This, of course, does not provide enough empirical data to make quantitative statements about the prevalence of excessive timestamp usage. However, the impression is strong that apps commonly implement activity timestamping without duly considering their privacy impact

and without regard for data minimisation. I can only assume that this is due to a lack of sensitivity and awareness on the side of developers, and a different focus of priorities on the side of personnel responsible for privacy compliance. In that sense, my work demonstrates that data minimisation should be at the heart of privacy by design. Before measures are taken to secure collected data or to process it more privacy-friendly, the preceding question for design and engineering should always be, whether collecting data like this is necessary in the first place. I hope that my work raises awareness for considering timestamp minimisation within app development and auditing.

Considering timestamps for data minimisation implies that developers and operators become more rigorous about specifying purposes as well. Collecting timestamps for unspecified purposes is clearly a privacy violation within GDPR. My work emphasises that the obligations of data minimisation and purpose specification have to be interpreted much further: Collecting high-precision timestamps should not be seen as legitimate if under some special conditions, e. g., for investigating misconduct, the high precision might be needed. I could showcase that timestamps processed for such conflated purposes should be separated and their minimisation potential should be evaluated separately.

The real-world application of this best practice is likely limited by software developers' objection that this induces undue cost and limits the flexibility of the use cases that their apps might serve. To that extend, I could showcase that the effort of transforming apps to minimised timestamps is very much manageable. Moreover, this thesis should hopefully help to make it clear, that avoiding minimisation for the sake of flexibility in use, just passes the burden of compliance onto their customer, the operator, who arguably does not have the means to make meaningful changes to the processing at this point anymore.

8.3.2 Protecting Mobile Workers

In my work, I could demonstrate that protecting the privacy of mobile workers on field service or business travel is not just a matter of installing a VPN on their company notebook. My experimental results make it clear that protecting against traffic leakage and activity exposure requires more effort. Our findings made it obvious that platform and VPN app manufacturers do not sufficiently test their security and privacy properties under real-world environments. Experimental security and privacy analyses like ours should be routinely performed as part of QA testing. Moreover, I tried to make a case for the responsibility of platform vendors to provide more misuse-resilient APIs that make it easier for VPN app manufacturers to securely interface with the platform. Closing this gap in leak-proof VPN is, however, limited by the platform vendors unwillingness to cater for these 'special' requirements of VPN apps, as was demonstrated by Apple's reaction to not include a blocking fail state in their VPN API. Nonetheless, I hope that my work can help to raise awareness for the intricacies of secure VPN establishment, and that just using ordinary VPN is not sufficient for protecting against observability of communication on the go.

8.3.3 Ubiquitous Identifiability and Attribution

Looking at real-world apps revealed that the complexity of their data models is a challenge for compliance documentation. Achieving documentation that can inform data subjects in a comprehensive and still comprehensible way requires additional measures that go beyond the distinctions defined by GDPR. I introduced two such distinctions on a semantic level: the concepts of a dedicated versus shared attributability and decisive roles. These distinctions are not grading whether or not data should be considered personal, but rather aim to highlight subsets of personal data that are particularly relevant for a given category of data subjects.

Legal scholars need to look into the problem of ubiquitous identifiability and assess whether such heuristic and structural relevance filtering approaches, like the one proposed in Chapter 7, are viable under the law. In any case, the problem of ubiquitous identifiability makes it obvious that legal guidance is necessary to clarify what constitutes a GDPR-compliant trade-off between comprehensive yet incomprehensible listings of personal data categories and succinct listings of vague umbrella terms. This is especially important, as controllers, without clear guidelines, arguably are intrinsically motivated towards vague wording, because vague umbrella terms naturally require less updating effort and better hide questionable practices from data subjects and auditors. In that sense, the impact of our approach to involve developers as domain experts is most likely limited by a separation of duty between legal data protection and software engineering that shies away from involving developers in compliance obligations. Moreover, software vendors might fear they assumed legal responsibility by providing compliance documentation with interpretive character. It would be up to policy makers to set the incentives for software vendors to cooperate more in compliance.

9 Outlook on Future Work

In most areas of my work, general research interest is still small, but hopefully growing. Within applied data protection, I found a niche in applied data minimisation and timestamp privacy. Some of my contributions in this niche have been the first of their kind, e. g., the case studies on timestamp usage and the empirical analysis of precision demand. Therefore, I would say that research on applied data minimisation is still at the beginning and that there are many open research questions when it comes to investigating minimisation potentials and best practices from a purpose-driven perspective. This includes tool support for developers and also figuring out privacy anti-patterns that stand in the way of more privacy-friendly app designs.

For a total account of user privacy, research has to also have an eye on the underlying infrastructure and how its flaws might undo privacy gains on the application level. I pointed this out for the observability of user activity due to insecure VPN establishment. But similar threats to activity exposure might also exist on lower levels at the communication endpoints, i. e., insufficient isolation at the user client or the server.

A vast field of open research topics could be found at the intersection of GDPR compliance documentation and applied data protection. I demonstrated for the compilation of personal data category lists how technical solutions can support compliance tasks. It also suggests how developers can be involved in compliance processes to not only make them more integrated and up to date, but hopefully also more truthful. Privacy research has investigated many ways to better convey privacy information to data subjects. But in the end, the underlying information has to be specific and comprehensive enough to allow an *informed* decision about whether a user finds the *actual* app behaviour acceptable. Otherwise, compliance information is pointless or even deceptive. Exploring more technical ways to derive compliance documentation from actual app behaviour could therefore lead to more honesty in privacy.

In the following, I want to go into more detail about some suggested areas of future work that tie to the contributions I presented in this dissertation.

9.1 Minimisation of Activity Timestamping

One focus of the future work on activity timestamp minimisation has to be on improving its operationalisation. As timestamping purposes and demands are highly application specific, figuring out programmatic purposes and determining necessary precision levels are tasks that have to be exercised over and over. The case studies presented in this dissertation used a rather labour-intensive method to determine programmatic uses of timestamps. One area of research could therefore be to explore ways to distinguish more automatically, what constitutes

syntactically and semantically relevant references to personal timestamps, i. e., code locations that are likely candidates of programmatic use. Another, approach for operationalisation could be to (again) involve the software developer in the documentation of programmatic uses. Through code annotations, developers could declare new uses and automatic QA tests could alert reviewers to undeclared uses during change management.

User studies also seem promising to help understanding how widespread excessive timestamping is among developers and what the underlying causes are. With respect to our proposed timestamp alternatives, user studies could also investigate if developers succeed at designing more data-minimal apps given our framework and what functionality they are missing. That said, additional timestamp alternatives could be investigated to further replace use cases of conventional or reduced-precision timestamps. As I alluded to in Chapter 3, cryptographic approaches might be a way to resolve the need for external referencing, e. g., for filtering or expiry usages.

In the introduction, I talked about our attempts to add direct user feedback to our estimation of precision demand for informative timestamps. Future work could follow up on our approach to learn about users' precision choices through a precision-reducing web extension. On a meta level, a challenge remains how studies can figure out the necessary precision without annoying participants.

9.2 Privacy Risk Public Wi-Fi

Public Wi-Fi networks will likely continue to play an important role in the privacy of mobile workers and Internet users in general. Since our work on Wi-Fi and VPN privacy was published, Mozilla has entered the market of Personal VPNs, with the explicit use case of protecting against malicious or privacy-invasive Wi-Fi providers [Moz22]. In fact, four out of the five situations given by Mozilla for why you should use a VPN involve malicious Wi-Fi. Moreover, Apple is beta testing a VPN-like solution called iCloud Private Relay that is also marketed as a solution to protect traffic data that 'can be seen by your network provider' [App22]. With these two big players pushing privacy measures against untrusted access networks, we can expect to see more attention on this topic.

Future work should continue to investigate how resilient new approaches are against disruptions to VPN establishment. Especially the handling of captive portals is becoming a weak point if Wi-Fi operators themselves are presumed adversarial, given that some form of interaction remains necessary for captive network remediation. In that sense, data exfiltration by captive portals should be investigated as a privacy risk. Another open topic could be, how comprehensively approaches like Apple's Privacy Relay can protect against activity exposure, considering that its scope is limited to browser traffic from Safari. From a methodological perspective, more research is needed to ensure more complete leakage tests. Our experiments clearly showed that protective measures are *not* uniformly applied by platforms, depending on the mechanism used for communication (e. g., high-level API vs. BSD sockets) and depending on who the

sender is (e. g., platform processes vs. third-party app). To that extent, approaches to improve the coverage of leak experiments are needed.

9.3 Integrated Data Protection Compliance

I am convinced that integrating data protection compliance tasks more closely into software development processes, like it has been previously done with QA and deployment, could drastically improve the quality and availability of compliance information. Future work could investigate how data subjects and auditors perceive information written by compliance personnel compared to information generated on the basis of developer annotations. It is essential that such research considers not only the comprehensibility of such information but also the perceived specificity and correctness.

Another open topic could be, how the ability to automate the compilation of, e. g., personal data categories can be used for change monitoring and QA. I presumed that comparing listings between changes could be a way to monitor for unintended personal data collection, or at least highlight data model changes that need a privacy-focused review. Case studies with historical change records could be done to assess the practicality of such alerts.

One further aspect of deriving compliance information from the source of truth, that I had to leave for future work, is the selection of appropriate umbrella terms used for summarising or abstracting multiple data model attributes. Although developers are the domain experts, they might find it challenging to come up with such terms. Moreover, legal guidelines on what constitutes good abstractions remain superficial (cf. Sect. 2.2). Future work could investigate how developers could be guided towards finding good umbrella terms. To some extent, the selection could even be assisted by providing automatically derived suggestions. Such suggestions could be generated with automatically generated hypernyms of the summarised attributes or associated UI text. Moreover, research is need on how a negotiation between legal and development could be conducted to find terms that are agreeable for developers with respect to factual correctness as well as agreeable for legal advisors with respect to their legal implications.

Bibliography

[Ali+19] Suzan Ali, Tousif Osman, Mohammad Mannan and Amr Youssef. *"On Privacy Risks of Public WiFi Captive Portals"*. In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Vol. 11737. Lecture Notes in Computer Science. Springer International Publishing, 2019, pp. 80–98. ISBN: 978-3-030-31500-9. DOI: 10.1007/978-3-030-31500-9_6.

[ALS15] Talayeh Aledavood, Sune Lehmann and Jari Saramäki. *"Digital Daily Cycles of Individuals"*. In: *Frontiers in Physics* 3 (2015). ISSN: 2296-424X. DOI: 10.3389/fphy.2015.00073.

[And] Android Developers. *"VPN"*. URL: https://developer.android.com/guide/topics/connectivity/vpn.

[And94] Martin Andersson. *"Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering"*. In: *Entity-Relationship Approach — ER '94 Business Modelling and Re-Engineering*. Red. by Gerhard Goos, Juris Hartmanis and Jan Leeuwen. Vol. 881. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1994, pp. 403–419. ISBN: 978-3-540-49100-2. DOI: 10.1007/3-540-58786-1_93.

[Ans+22] Johanna Ansohn McDougall, Christian Burkert, Daniel Demmler, Monina Schwarz, Vincent Hubbe and Hannes Federrath. *"Probing for Passwords – Privacy Implications of SSIDs in Probe Requests"*. In: *Applied Cryptography and Network Security*. Lecture Notes in Computer Science. Springer International Publishing, 2022, pp. 376–395. ISBN: 978-3-031-09234-3. DOI: 10.1007/978-3-031-09234-3_19.

[Appa] Apple. *"NEOnDemandRuleConnect: A VPN On Demand rule that connects the VPN"*. URL: https://developer.apple.com/documentation/networkextension/neondemandruleconnect.

[Appb] Apple. *"Packet Tunnel Provider"*. URL: https://developer.apple.com/documentation/networkextension/packet_tunnel_provider.

[Appc] Apple. *"Personal VPN"*. URL: https://developer.apple.com/documentation/networkextension/personal_vpn.

[Appd] Apple. *"VPN On Demand Rules"*. URL: https://developer.apple.com/documentation/networkextension/personal_vpn/vpn_on_demand_rules.

[App22] Apple. *"About iCloud Private Relay"*. Apple Support. 2022. URL: https://support.apple.com/en-us/HT212614.

[Art13] Article 29 Working Party. *"Opinion 03/2013 on Purpose Limitation (WP203)"*. 2013.

[Art18] Article 29 Working Party. *"Guidelines on Transparency under Regulation 2016/679 (Wp260rev.01)"*. 11th Apr. 2018.

[BAF22] Christian Burkert, Johanna Ansohn McDougall and Hannes Federrath. *"Data Minimisation Potential for Timestamps in Git: An Empirical Analysis of User Configurations"*. In: *ICT Systems Security and Privacy Protection*. IFIP Advances in Information and Communication Technology. Springer International Publishing, 2022, pp. 323–339. ISBN: 978-3-031-06975-8. DOI: 10.1007/978-3-031-06975-8_19.

[BBF21] Christian Burkert, Maximilian Blochberger and Hannes Federrath. *"Compiling Personal Data and Subject Categories from App Data Models"*. In: *ICT Systems Security and Privacy Protection*. Vol. 625. IFIP Advances in Information and Communication Technology. Springer International Publishing, 2021, pp. 242–255. ISBN: 978-3-030-78120-0. DOI: 10.1007/978-3-030-78120-0_16.

[BBF22] Christian Burkert, Jonathan Balack and Hannes Federrath. *"PrivacyDates: A Framework for More Privacy-Preserving Timestamp Data Types"*. In: Sicherheit 2022: Sicherheit, Schutz und Zuverlässigkeit: Konferenzband der 11. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI), 5.-8. April 2022 in Karlsruhe. GI-Edition. Proceedings Volume P-323. Gesellschaft für Informatik e.V. (GI), 2022. ISBN: 978-3-88579-717-3.

[BF19] Christian Burkert and Hannes Federrath. *"Towards Minimising Timestamp Usage In Application Software: A Case Study of the Mattermost Application"*. In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Vol. 11737. Lecture Notes in Computer Science. Springer International Publishing, 2019, pp. 138–155. ISBN: 978-3-030-31500-9. DOI: 10.1007/978-3-030-31500-9_9.

[BG09] B. Bercic and C. George. *"Identifying Personal Data Using Relational Database Design Principles"*. In: *International Journal of Law and Information Technology* 17.3 (1st Sept. 2009), pp. 233–251. ISSN: 0967-0769, 1464-3693. DOI: 10.1093/ijlit/ean007.

[BH17] Felix Bieker and Marit Hansen. *"Normen des technischen Datenschutzes nach der europäischen Datenschutzreform"*. In: *Datenschutz und Datensicherheit - DuD* 41.5 (1st May 2017), pp. 285–289. ISSN: 1862-2607. DOI: 10.1007/s11623-017-0776-1.

[Blo+19] Maximilian Blochberger, Jakob Rieck, Christian Burkert, Tobias Mueller and Hannes Federrath. *"State of the Sandbox: Investigating macOS Application Security"*. In: *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*. WPES'19. Association for Computing Machinery, 11th Nov. 2019, pp. 150–161. ISBN: 978-1-4503-6830-8. DOI: 10.1145/3338498.3358654.

[Böh67] Hermann Böhrs. *"Arbeitsstudien in der Betriebswirtschaft"*. Gabler Verlag, 1967. ISBN: 978-3-663-04207-5. DOI: 10.1007/978-3-663-04207-5.

[Bor18] Stephanie Bornstein. *"Antidiscriminatory Algorithms"*. In: *Alabama Law Review* 70.2 (2018), p. 519.

[Bur+21] Christian Burkert, Johanna Ansohn McDougall, Hannes Federrath and Mathias Fischer. *"Analysing Leakage during VPN Establishment in Public Wi-Fi Networks"*. In: *ICC 2021 - IEEE International Conference on Communications*. ICC 2021 - IEEE International Conference on Communications. IEEE, June 2021, pp. 1–6. ISBN: 978-1-72817-122-7. DOI: 10.1109/ICC42927.2021.9500375.

[Bur22] Christian Burkert. *".gitconfig Date Study Dataset"*. 2022. URL: https://github.com/EMPRI-DEVOPS/gitconfig-study-dataset.

[Che+13] Ningning Cheng, Xinlei Oscar Wang, Wei Cheng, Prasant Mohapatra and Aruna Seneviratne. *"Characterizing Privacy Leakage of Public WiFi Networks for Users on Travel"*. In: *2013 Proceedings IEEE INFOCOM*. IEEE INFOCOM 2013 - IEEE Conference on Computer Communications. IEEE, Apr. 2013, pp. 2769–2777. ISBN: 978-1-4673-5946-7.

[Cla+18] Maëlick Claes, Mika V. Mäntylä, Miikka Kuutila and Bram Adams. *"Do Programmers Work at Night or During the Weekend?"* In: ICSE. ACM, 2018. ISBN: 978-1-4503-5638-1. DOI: 10.1145/3180155.3180193.

[Col+19] Michael Colesky, Jaap-Henk Hoepman, Christoph Boesch, Frank Kargl, Henning Kopp, Patrick Mosby, Daniel Le Métayer, Olha Drozd, José M. del Álamo, Yod Samuel Martín, Julio C. Caiza, Mohit Gupta and Nick Doty. *"privacypatterns.org"*. 2019. URL: https://privacypatterns.org.

[Dam16] Ulrich Dammann. *"Erfolge und Defizite der EU-Datenschutzgrundverordnung – Erwarteter Fortschritt, Schwächen und überraschende Innovationen"*. In: *ZD Zeitschrift für Datenschutz* 7 (2016), pp. 307–314.

[Dan+15] George Danezis, Josep Domingo-Ferrer, Marit Hansen, Jaap-Henk Hoepman, Daniel Le Métayer, Rodica Tirtea and Stefan Schiffner. *"Privacy and Data Protection by Design - from policy to engineering"*. In: *CoRR* abs/1501.03726 (2015).

[Dat18] Datenschutzkonferenz (DSK). *"Hinweise zum Verzeichnis von Verarbeitungstätigkeiten, Art. 30 DS-GVO"*. Feb. 2018.

[DF89] Yvo Desmedt and Yair Frankel. *"Threshold Cryptosystems"*. In: *CRYPTO*. Vol. 435. Lecture Notes in Computer Science. Springer, 1989, pp. 307–315.

[DiC19] Michael DiClaudio. *"People analytics and the rise of HR: how data, analytics and emerging technology can transform human resources (HR) into a profit center"*. In: *Strategic HR Review* 18.2 (13th Feb. 2019), pp. 42–46. ISSN: 1475-4398. DOI: 10.1108/SHR-11-2018-0096.

[DM12] Primavera De Filippi and Smari McCarthy. *"Cloud Computing: Centralization and Data Sovereignty"*. SSRN Scholarly Paper. 26th Oct. 2012.

[Dra+19] Kostas Drakonakis, Panagiotis Ilia, Sotiris Ioannidis and Jason Polakis. *"Please Forget Where I Was Last Summer: The Privacy Risks of Public Location (Meta)Data"*. In: *NDSS*. Jan. 2019. DOI: 10.14722/ndss.2019.23151.

[EMP] EMPRI-DEVOPS. *"git-privacy"*. URL: https://github.com/EMPRI-DEVOPS/git-privacy.

[EMP22a] EMPRI-DEVOPS. *"django-privacydates"*. 2022. URL: https://github.com/EMPRI-DEVOPS/django-privacydates.

[EMP22b] EMPRI-DEVOPS. *"GitHub Privacy"*. 2022. URL: https://github.com/EMPRI-DEVOPS/empri-browser-extension.

[ETL11] Jon Eyolfson, Lin Tan and Patrick Lam. *"Do Time of Day and Developer Experience Affect Commit Bugginess?"* In: *MSR*. ACM, 2011. ISBN: 978-1-4503-0574-7. DOI: 10.1145/1985441.1985464.

[Eur20] European Data Protection Board. *"Guidelines 4/2019 on Article 25 Data Protection by Design and by Default - European Data Protection Board"*. 20th Oct. 2020.

[FCC11] Georgios John Fakas, Ben Cawley and Zhi Cai. *"Automated Generation of Personal Data Reports from Relational Databases"*. In: *Journal of Information & Knowledge Management* 10.02 (June 2011), pp. 193–208. ISSN: 0219-6492, 1793-6926. DOI: 10.1142/S0219649211002936.

[FR14] Roy T. Fielding and Julian Reschke. *"Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests"*. RFC 7232. June 2014. DOI: 10.17487/RFC7232.

[Gam21] Game World Observer. *"Xsolla cites growth rate slowdown as reason for layoffs, CEO's tweet causes further controversy"*. 5th Aug. 2021. URL: https://gameworldobserver.com/?p=10949.

[GBW16] Sandra Greiner, Thomas Buchmann and Bernhard Westfechtel. *"Bidirectional Transformations with QVT-R: A Case Study in Round-Trip Engineering UML Class Models and Java Source Code"*. In: *MODELSWARD 2016 - Proceedings of the 4rd International Conference on Model-Driven Engineering and Software Development, Rome, Italy, 19-21 February, 2016*. SciTePress, 2016, pp. 15–27. DOI: 10.5220/0005644700150027.

[Git21a] Git. *"Git Source Code"*. 2021. URL: https://github.com/git/git.

[Git21b] GitHub Docs. *"Best practices for integrators"*. 2021. URL: https://docs.github.com/en/rest/guides/best-practices-for-integrators.

[Git21c] GitHub Docs. *"Search API"*. 2021. URL: https://docs.github.com/en/rest/reference/search.

[Git22] Git. *"Reference"*. 2022. URL: https://git-scm.com/docs.

[Goo19a] Google, Inc. *"Go testing package"*. 2019. URL: https://golang.org/pkg/testing/.

[Goo19b] Google, Inc. *"Go Tools gorename command"*. 2019. URL: https://godoc.org/golang.org/x/tools/cmd/gorename.

[Gou13] Georgios Gousios. *"The GHTorrent dataset and tool suite"*. In: MSR '13. 2013. ISBN: 978-1-4673-2936-1.

[Hoe14] Jaap-Henk Hoepman. *"Privacy Design Strategies"*. In: *SEC*. Vol. 428. IFIP Advances in Information and Communication Technology. Springer, 2014, pp. 446–459.

[HTM19] Dominik Huth, Ahmet Tanakol and Florian Matthes. *"Using Enterprise Architecture Models for Creating the Record of Processing Activities (Art. 30 GDPR)"*. In: *2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC)*. 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC). IEEE, Oct. 2019, pp. 98–104. ISBN: 978-1-72812-702-6. DOI: 10.1109/EDOC.2019.00021.

[Ikr+16] Muhammad Ikram, Narseo Vallina-Rodriguez, Suranga Seneviratne, Mohamed Ali Kaafar and Vern Paxson. *"An Analysis of the Privacy and Security Risks of Android VPN Permission-Enabled Apps"*. In: *Proceedings of the 2016 ACM on Internet Measurement Conference - IMC '16*. The 2016 ACM. ACM Press, 2016, pp. 349–364. ISBN: 978-1-4503-4526-2. DOI: 10.1145/2987443.2987471.

[Kam+07] Pandurang Kamat, Wenyuan Xu, Wade Trappe and Yanyong Zhang. *"Temporal Privacy in Wireless Sensor Networks"*. In: *27th IEEE International Conference on Distributed Computing Systems (ICDCS 2007), June 25-29, 2007, Toronto, Ontario, Canada*. IEEE Computer Society, 2007, p. 23. DOI: 10.1109/ICDCS.2007.146.

[Kar+19] Frank Kargl et al. *"privacypatterns.eu"*. 2019. URL: https://privacypatterns.eu.

[Kar17] Rickard Karlsson. *"EzMole: A new prototype for securing public Wi-Fi connections"*. MA thesis. Luleå University of Technology, Computer Science, 2017, p. 58.

[KB20] Jürgen Kühling and Benedikt Buchner. *"Datenschutz-Grundverordnung, Bundesdatenschutzgesetz – Kommentar"*. 3. Auflage. C.H. Beck, 2020. 1877 pp. ISBN: 978-3-406-74994-0.

[Ker07] Florian Kerschbaum. *"Distance-preserving pseudonymization for timestamps and spatial data"*. In: *Proceedings of the 2007 ACM Workshop on Privacy in the Electronic Society, WPES 2007, Alexandria, VA, USA, October 29, 2007*. ACM, 2007, pp. 68–71.

[KGV18] Dimitrios Karapiperis, Aris Gkoulalas-Divanis and Vassilios S. Verykios. *"FEDERAL: A Framework for Distance-Aware Privacy-Preserving Record Linkage"*. In: *IEEE Transactions on Knowledge and Data Engineering* 30.2 (2018), pp. 292–304.

[Kha+18] Mohammad Taha Khan, Joe DeBlasio, Geoffrey M. Voelker, Alex C. Snoeren, Chris Kanich and Narseo Vallina-Rodriguez. *"An Empirical Analysis of the Commercial VPN Ecosystem"*. In: *Proceedings of the Internet Measurement Conference 2018*. IMC '18: Internet Measurement Conference. ACM, 31st Oct. 2018, pp. 443–456. ISBN: 978-1-4503-5619-0. DOI: 10.1145/3278532.3278570.

[Kum+15] Warren "Ace" Kumari, Ólafur Guðmundsson, P Ebersman and Steve Sheng. *"Captive-Portal Identification Using DHCP or Router Advertisements (RAs)"*. RFC 7710. Dec. 2015. DOI: 10.17487/RFC7710.

[Kun+20] *"The EU General Data Protection Regulation (GDPR): A Commentary"*. Oxford University Press, 2020. 1488 pp. ISBN: 978-0-19-882649-1. DOI: 10.1093/oso/9780198826491.001.0001.

[LFH17] Jörg Lenhard, Lothar Fritsch and Sebastian Herold. *"A Literature Study on Privacy Patterns Research"*. In: *43rd Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2017, Vienna, Austria, August 30 - Sept. 1, 2017*. IEEE Computer Society, 2017, pp. 194–201. DOI: 10.1109/SEAA.2017.28.

[Mar+18a] Matthias Marx, Maximilian Blochberger, Christian Burkert, Dominik Herrmann and Hannes Federrath. *"Privatsphäre als inhärente Eigenschaft eines Kommunikationsnetzes am Beispiel einer Anonymisierungslösung für IPv6"*. In: *Die Fortentwicklung des Datenschutzes: Zwischen Systemgestaltung und Selbstregulierung*. DuD-Fachbeiträge. Springer Fachmedien, 2018, pp. 209–223. ISBN: 978-3-658-23727-1. DOI: 10.1007/978-3-658-23727-1_12.

[Mar+18b] Matthias Marx, Erik Sy, Christian Burkert and Hannes Federrath. *"Anonymity Online – Current Solutions and Challenges"*. In: *Privacy and Identity Management. The Smart Revolution: 12th IFIP WG 9.2, 9.5, 9.6/11.7, 11.6/SIG 9.2.2 International Summer School, Ispra, Italy, September 4-8, 2017, Revised Selected Papers*. Springer International Publishing, 2018, pp. 38–55. ISBN: 978-3-319-92925-5. DOI: 10.1007/978-3-319-92925-5_4.

[Mar19] MariaDB. *"Data Type Storage Requirements"*. Apr. 2019. URL: https://mariadb.com/kb/en/data-type-storage-requirements/.

[Mat] Mattermost, Inc. *"Mattermost Website"*. URL: https://www.mattermost.org.

[Mat18a] Mattermost, Inc. *"Mattermost Server v4.8.0"*. 2018. URL: https://github.com/mattermost/mattermost-server/releases/tag/v4.8.0.

[Mat18b] Mattermost, Inc. *"Mattermost Webapp v5.5.1"*. 2018. URL: https://github.com/mattermost/mattermost-webapp/releases/tag/v5.5.1.

[Mic] Microsoft. *"Workplace Analytics"*. URL: https://products.office.com/en-us/business/workplace-analytics.

[MK18a] Yod-Samuel Martin and Antonio Kung. *"Methods and Tools for GDPR Compliance Through Privacy and Data Protection Engineering"*. In: *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). IEEE, Apr. 2018, pp. 108–111. ISBN: 978-1-5386-5445-3. DOI: 10.1109/EuroSPW.2018.00021.

[MK18b] Paola Mavriki and Maria Karyda. *"Profiling with big data: Identifying privacy implications for idividuals, groups and society"*. In: *MCIS*. Oct. 2018.

[Moz22] Mozilla. *"5 Reasons You Should Use a VPN"*. 2022. URL: https://www.mozilla.org/en-US/products/vpn/more/when-to-use-a-vpn/.

[MR18] Shane McCulley and Vassil Roussev. *"Latent Typing Biometrics in Online Collaboration Services"*. In: *Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC 2018, San Juan, PR, USA, December 03-07, 2018*. ACM, 2018, pp. 66–76. DOI: 10.1145/3274694.3274754.

[MVS21] João Eduardo Montandon, Marco Tulio Valente and Luciana L. Silva. *"Mining the Technical Roles of GitHub Users"*. In: *Information and Software Technology* 131 (1st Mar. 2021), p. 106485. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2020.106485.

[NNH04] Flemming Nielson, Hanne R. Nielson and Chris Hankin. *"Principles of Program Analysis"*. Springer Science & Business Media, 10th Dec. 2004. 482 pp. ISBN: 978-3-540-65410-0. Google Books: RLjt0xSj8DcC.

[Ogr17] Claudia Ogriseg. *"GDPR and Personal Data Protection in the Employment Context"*. In: *Labour & Law Issues* 3.2 (14th Dec. 2017), pp. 1–24. ISSN: 2421-2695. DOI: 10.6092/issn.2421-2695/7573.

[OHM13] Saya Onoue, Hideaki Hata and Ken-ichi Matsumoto. *"A Study of the Characteristics of Developers' Activities in GitHub"*. In: *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*. Vol. 2. 2013, pp. 7–12. DOI: 10.1109/APSEC.2013.104.

[Pan14] Vijay Pandurangan. *"On Taxis and Rainbows. Lessons from NYC's improperly anonymized taxi logs"*. 21st June 2014. URL: https://tech.vijayp.ca/of-taxis-and-rainbows-f6bc289679a1.

[Pat+20] Madushi H. Pathmaperuma, Yogachandran Rahulamathavan, Safak Dogan and Ahmet M. Kondoz. *"In-App Activity Recognition from Wi-Fi Encrypted Traffic"*. In: *Intelligent Computing*. Advances in Intelligent Systems and Computing. Springer International Publishing, 2020, pp. 685–697. ISBN: 978-3-030-52249-0. DOI: 10.1007/978-3-030-52249-0_46.

[Per+15] Vasile C. Perta, Marco V. Barbera, Gareth Tyson, Hamed Haddadi and Alessandro Mei. *"A Glance through the VPN Looking Glass: IPv6 Leakage and DNS Hijacking in Commercial VPN clients"*. In: *Proceedings on Privacy Enhancing Technologies* 2015.1 (1st Apr. 2015), pp. 77–91. ISSN: 2299-0984. DOI: 10.1515/popets-2015-0006.

[PMB14] AJ Paverd, Andrew Martin and Ian Brown. *"Modelling and automatically analysing privacy properties for honest-but-curious adversaries"*. Tech. rep. 2014.

[Pro20] Proton Team. *"VPN Bypass Vulnerability in Apple iOS"*. 25th Mar. 2020. URL: https://protonvpn.com/blog/apple-ios-vulnerability-disclosure/.

[Pyt21] Python Software Foundation. *"datetime - Basic date and time types"*. May 2021. URL: https://docs.python.org/3/library/datetime.html.

[Rei+16] Joel R. Reidenberg, Jaspreet Bhatia, Travis D. Breaux and Thomas B. Norton. *"Ambiguity in Privacy Policies and the Impact of Regulation"*. In: *The Journal of Legal Studies* 45.S2 (June 2016), S163–S190. ISSN: 0047-2530. DOI: 10.1086/688669.

[Res08] Pete Resnick. *"Internet Message Format"*. RFC 5322. Oct. 2008. DOI: 10.17487/RFC5322.

[RN16] Ayushi Rastogi and Nachiappan Nagappan. *"On the Personality Traits of GitHub Contributors"*. In: *27th IEEE International Symposium on Software Reliability Engineering, ISSRE 2016, Ottawa, ON, Canada, October 23-27, 2016*. IEEE Computer Society, 2016, pp. 77–86. DOI: 10.1109/ISSRE.2016.43.

[Roi17] Antoni Roig. *"Safeguards for the right not to be subject to a decision based solely on automated processing (Article 22 GDPR)"*. In: *European Journal of Law and Technology* 8.3 (2017).

[RPB20] Paul Ryan, Harshvardhan J. Pandit and Rob Brennan. *"A Common Semantic Model of the GDPR Register of Processing Activities"*. In: *Legal Knowledge and Information Systems - JURIX 2020: The Thirty-Third Annual Conference, Brno, Czech Republic, December 9-11, 2020*. Vol. 334. Frontiers in Artificial Intelligence and Applications. IOS Press, 2020, pp. 251–254. DOI: 10.3233/FAIA200876.

[SA18] Awanthika Senarath and Nalin Asanka Gamagedara Arachchilage. *"Understanding Software Developers' Approach towards Implementing Data Minimization"*. 2018. DOI: 10.48550/ARXIV.1808.01479. URL: https://arxiv.org/abs/1808.01479.

[SH03] Abigail J. Sellen and Richard H. R. Harper. *"The Myth of the Paperless Office"*. MIT Press, 28th Feb. 2003. 246 pp. ISBN: 978-0-262-25049-8. Google Books: GhbvCwAAQBAJ.

[SHS19] Spiros Simitis, Gerrit Hornung and Indra Spiecker Döhmann. *"Datenschutzrecht: DSGVO mit BDSG"*. 1. Auflage. NomosKommentar. Nomos, 2019. 1474 pp. ISBN: 978-3-8487-3590-7.

[SLL06] Adam J. Slagell, Kiran Lakkaraju and Katherine Luo. *"FLAIM: A Multi-level Anonymization Framework for Computer and Network Logs"*. In: *LISA*. USENIX, 2006, pp. 63–77.

[Som+19] Nissy Sombatruang, Lucky Onwuzurike, M. Angela Sasse and Michelle Baddeley. *"Factors influencing users to use unsecured wi-fi networks: evidence in the wild"*. In: *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*. WiSec '19: 12th ACM Conference on Security and Privacy in Wireless and Mobile Networks. ACM, 15th May 2019, pp. 203–213. ISBN: 978-1-4503-6726-4. DOI: 10.1145/3317549.3323412.

[Swe02] Latanya Sweeney. *"k-Anonymity: A Model for Protecting Privacy"*. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.5 (2002), pp. 557–570. DOI: 10.1142/S0218488502001648.

[Sy+18] Erik Sy, Christian Burkert, Hannes Federrath and Mathias Fischer. *"Tracking Users across the Web via TLS Session Resumption"*. In: *Proceedings of the 34th Annual Computer Security Applications Conference*. ACSAC '18. Association for Computing Machinery, 3rd Dec. 2018, pp. 289–299. ISBN: 978-1-4503-6569-7. DOI: 10.1145/3274694.3274708.

[Sy+19a] Erik Sy, Christian Burkert, Hannes Federrath and Mathias Fischer. *"A QUIC Look at Web Tracking"*. In: *Proceedings on Privacy Enhancing Technologies* 2019.3 (1st July 2019), pp. 255–266. ISSN: 2299-0984. DOI: 10.2478/popets-2019-0046.

[Sy+19b] Erik Sy, Christian Burkert, Tobias Mueller, Hannes Federrath and Mathias Fischer. *"QUICker Connection Establishment with Out-Of-Band Validation Tokens"*. In: *2019 IEEE 44th Conference on Local Computer Networks (LCN)*. 2019 IEEE 44th Conference on Local Computer Networks (LCN). Oct. 2019, pp. 105–108. DOI: 10.1109/LCN44214.2019.8990785.

[Sy+20] Erik Sy, Tobias Mueller, Christian Burkert, Hannes Federrath and Mathias Fischer. *"Enhanced Performance and Privacy for TLS over TCP Fast Open"*. In: *Proceedings on Privacy Enhancing Technologies* 2020.2 (1st Apr. 2020), pp. 271–287. DOI: 10.2478/popets-2020-0027.

[Tai21a] Taiga Agile. *"Taiga"*. Apr. 2021. URL: https://www.taiga.io/.

[Tai21b] Taiga Agile. *"taiga-back"*. Mar. 2021. URL: https://github.com/taigaio/taiga-back/releases/tag/6.0.7.

[Tai21c] Taiga Agile. *"taiga-front"*. Mar. 2021. URL: https://github.com/taigaio/taiga-front/releases/tag/6.0.7.

[TD18] Benjamin Traullé and Jean-Michel Dalle. *"The Evolution of Developer Work Rhythms: An Analysis Using Signal Processing Techniques"*. In: *Social Informatics*. Vol. 11185. 2018. DOI: 10.1007/978-3-030-01129-1_26.

[TG22] Jürgen Taeger and Detlev Gabel. *"DSGVO - BDSG - TTDSG"*. 4. Auflage. Fachmedien Recht und Wirtschaft, 2022. ISBN: 978-3-8005-9420-7.

[TLP18] Aizhan Tursunbayeva, Stefano Di Lauro and Claudia Pagliari. *"People analytics - A scoping review of conceptual boundaries and value propositions"*. In: *Int J. Information Management* 43 (2018), pp. 224–247. DOI: 10.1016/j.ijinfomgt.2018.08.002.

[tThi+14] M.C ten Thij, P Kampstra, S Bhulai, F Laux, P.M Pardalos and A Crolotte. *"Circadian Patterns in Twitter"*. In: IARIA, 2014. ISBN: 978-1-61208-358-2.

[Ven+13] Rahul Venkataramani, Atul Gupta, Allahbaksh Asadullah, Basavaraju Muddu and Vasudev Bhat. *"Discovery of Technical Expertise from Open Source Code Repositories"*. In: *Proceedings of the 22Nd International Conference on World Wide Web*. WWW '13 Companion. ACM, 2013, pp. 97–98. ISBN: 978-1-4503-2038-2. DOI: 10.1145/2487788.2487832.

[VFS13] Bogdan Vasilescu, Vladimir Filkov and Alexander Serebrenik. *"StackOverflow and GitHub: Associations between Software Development and Crowdsourced Knowledge"*. In: *2013 International Conference on Social Computing*. 2013 International Conference on Social Computing. Sept. 2013, pp. 188–195. DOI: 10.1109/SocialCom.2013.35.

[Wer+14] Marius Wernke, Pavel Skvortsov, Frank Dürr and Kurt Rothermel. *"A classification of location privacy attacks and approaches"*. In: *Personal and Ubiquitous Computing* 18.1 (2014), pp. 163–175.

[Wik20] Wikipedia. *"Captive Portal"*. 13th Mar. 2020. URL: https://en.wikipedia.org/wiki/Captive_portal.

[WK19] Nan Wang and Evangelos Katsamakas. *"A Network Data Science Approach to People Analytics"*. In: *Information Resources Management Journal* 32.2 (2019), pp. 28–51. DOI: 10.4018/IRMJ.2019040102.

[WMB20] Jack Wilson, David McLuskie and Ethan Bayne. *"Investigation into the Security and Privacy of iOS VPN Applications"*. In: *Proceedings of the 15th International Conference on Availability, Reliability and Security*. ARES '20. Association for Computing Machinery, 2020. ISBN: 978-1-4503-8833-7. DOI: 10.1145/3407023.3407029.

[Wol21] Wolfie Christl. *"Digitale Überwachung und Kontrolle am Arbeitsplatz"*. Cracked Labs, Wien, Sept. 2021.

[WZ19] Ian Wright and Albert Ziegler. *"The Standard Coder: A Machine Learning Approach to Measuring the Effort Required to Produce Source Code Change"*. In: *RAISE*. 2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE). May 2019. ISBN: 978-1-72812-272-4. DOI: 10.1109/RAISE.2019.00009.

[Yan+15] Xinyu Yang, Xuebin Ren, Shusen Yang and Julie McCann. *"A novel temporal perturbation based privacy-preserving scheme for real-time monitoring systems"*. In: *Computer Networks* 88 (2015), pp. 72–88. ISSN: 13891286. DOI: 10.1016/j.comnet.2015.06.007.

[ZBF21] Ephraim Zimmer, Christian Burkert and Hannes Federrath. *"Insiders Dissected: New Foundations and a Systematisation of the Research on Insiders"*. In: *Digital Threats: Research and Practice (DTRAP)* (22nd Oct. 2021). DOI: 10.1145/3473674.

[ZBY06] Jianqing Zhang, Nikita Borisov and William Yurcik. *"Outsourcing Security Analysis with Anonymized Logs"*. In: *Second International Conference on Security and Privacy in Communication Networks and the Workshops, SecureComm 2006, Baltimore, MD, USA, August 2, 2006 - September 1, 2006*. IEEE, 2006, pp. 1–9. DOI: 10.1109/SECCOMW.2006.359577.

[Zim+20] Ephraim Zimmer, Christian Burkert, Tom Petersen and Hannes Federrath. *"PEEPLL: Privacy-Enhanced Event Pseudonymisation with Limited Linkability"*. In: *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. Association for Computing Machinery, 30th Mar. 2020, pp. 1308–1311. ISBN: 978-1-4503-6866-7.

Eidesstattliche Versicherung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg, den 11. Oktober 2022

Christian Burkert