Towards Minimising Timestamp Usage in Application Software

A Case Study of the Mattermost Application

Christian Burkert and Hannes Federrath

University of Hamburg
{burkert,federrath}@informatik.uni-hamburg.de

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.

Keywords: Privacy by design · Data minimisation · Timestamps

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* [6, 30]. 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 [21].

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 [1]. In fact, the EU General Data Protection Regulation (GDPR) is also applicable in the context of employment [22]. 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 [4], 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 [29] and contribute to the linkability and de-anonymisation of data [32]. 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 [24]. 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 [20].

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 [19].

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 2 provides background on our adversary model. Section 3 analyses the usage of timestamps in Mattermost. Section 4 presents alternative design patterns. Section 5 discusses related work and Sect. 6 concludes the paper.

2 Adversary Model

Our adversary model follows the established honest-but-curious (HBC) notion commonly used to assess communication protocols. Paverd, Martin and Brown [25] 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. 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 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 [17]. To determine which timestamps are presented to the user, we used Mattermost Web Client in version 5.5.1 [18].

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.

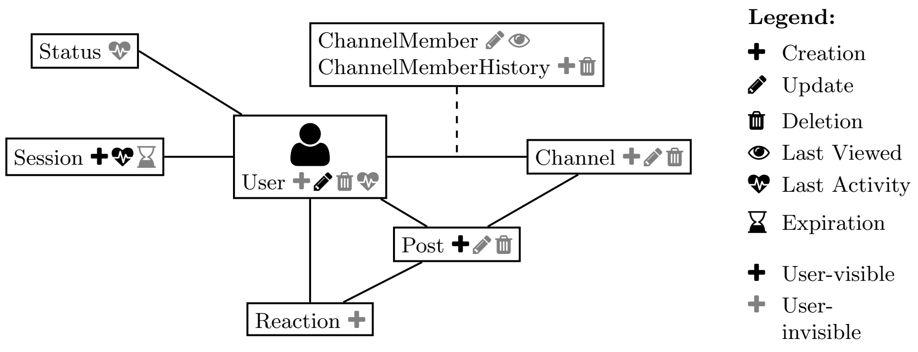

Figure 1. Mattermost's core components with their respective timestamps, categorized according to their type and visibility.

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` [9]. 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.

Table 1. 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.

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

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 1 lists

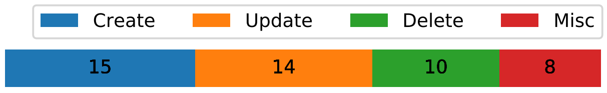

Figure 2. Distribution of timestamp types among the PII timestamps.

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.

3.2 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 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. 3.1 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 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. 3.1, 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 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).

3.4 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 [18]. 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.

3.5 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 [10]. 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.

Table 2. Types of use of PII timestamps within Mattermost's server code. Usage types that we consider as programmatic are highlighted by a grey background.

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

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 2. Following this basic classification, we assessed for each usage type whether or not it constitutes a programmatic use.

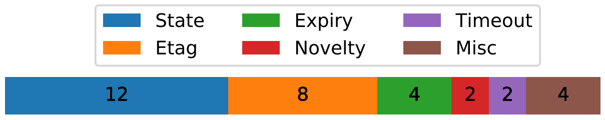

Figure 3. Distribution of programmatic uses between the identified usage types.

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 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 2 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 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.

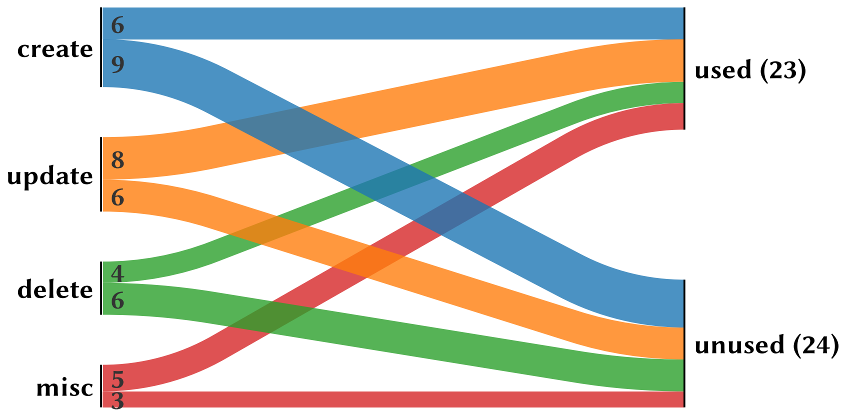

Figure 4. Distribution of timestamps types between used and unused timestamps.

Our investigation also showed that 24 out of the 47 PII timestamps have no programmatic use at all. Figure 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.

3.6 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).

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.

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 $reduce\colon \mathbb{N} \to \mathbb{N}$ 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 $\mathbb{T}_r := \big\{ reduce(t, r) \mid t \in \mathbb{N} \big\}$.*

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

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.

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.

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.

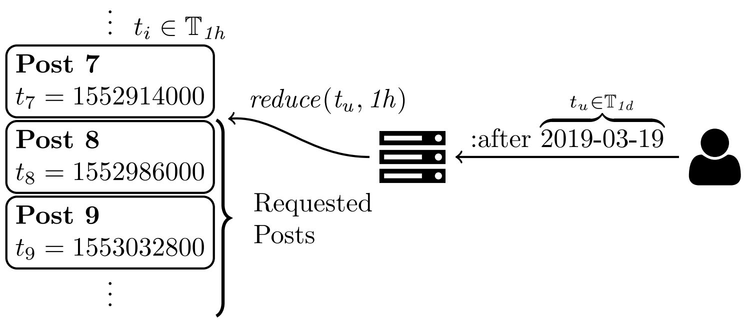

Figure 5. Filtering can be run on timestamps with significantly reduced precision. This example illustrates filters with a one-day resolution that are applied on post timestamps with a resolution of one hour.

Instead, we propose using per-post timestamps with a reduced resolution. Figure 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.

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 [8] 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 [8] 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.

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.

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.

4.8 User Information

Mattermost presents some timestamps to the user in its graphical user interface (see Fig. 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 timestamped 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 timestamp 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.

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

Table 3. Overview of timestamp usage and the respectively suited design alternatives.

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

cases, the potential for minimising PII timestamps is limited and depends on specific regulations. However, the 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 [5] in order to separate the duty of decryption among multiple parties for the sake of preventing misuse.

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 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). 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 extend 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.

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.

5.1 Timestamp Privacy

Basic redaction techniques for timestamps such as reduction and replacement with order-preserving counters have been presented by [28, 34] in the context of log anonymisation. Kerschbaum [15] 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 [12]. Countermeasures in that area include buffering to break the temporal correlation of creation and observation [12] and temporal perturbation by adding Laplace noise [33].

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 [13]. However, this approach also requires a trusted third party other than the data-custodians to achieve its privacy guarantees.

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, Tan and Lam [7] show that late-night contributions of software developers are statistically more buggy than contributions in the morning or during the day. Claes et al. [2] 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 [26] or to characterise them more generally [23]. 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 [6]. While PA is a trending topic, empirical evidence for its benefits or even concrete metrics are scarce [30]. 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 [31]. 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 [1] 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 [22], especially regarding automatic decision making [27].

5.3 Privacy Engineering

One of the privacy design strategies postulated by Hoepman [11] 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, the more privacy-preserving design alternatives presented in Sect. 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 [16]. 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 [3, 14], 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.

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

1. Bornstein, S.: Antidiscriminatory Algorithms. Alabama Law Review 70(2), 519 (2018)
2. Claes, M. et al.: Do programmers work at night or during the weekend? In: ICSE, pp. 705–715. ACM (2018)
3. Colesky, M. et al.: privacypatterns.org, (2019). `https://privacypatterns.org` (visited on 29th Mar. 2019)
4. Danezis, G. et al.: Privacy and Data Protection by Design - from policy to engineering. CoRR abs/1501.03726 (2015)
5. Desmedt, Y., and Frankel, Y.: Threshold Cryptosystems. In: CRYPTO. LNCS, vol. 435, pp. 307–315. Springer, Heidelberg (1989)
6. DiClaudio, M.: People analytics and the rise of HR: how data, analytics and emerging technology can transform human resources (HR) into a profit center. Strategic HR Review 18(2), 42–46 (2019)
7. Eyolfson, J., Tan, L., and Lam, P.: Do time of day and developer experience affect commit bugginess. In: Proceedings of the 8th International Working Conference on Mining Software Repositories, MSR 2011 (Co-located with ICSE), pp. 153–162. ACM (2011)
8. Fielding, R.T., and Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests. RFC 7232, (2014)
9. Google, Inc: Go testing package, (2019). `https://golang.org/pkg/testing/` (visited on 1st Mar. 2019)
10. Google, Inc: Go Tools gorename command, (2019). `https://godoc.org/golang.org/x/tools/cmd/gorename` (visited on 4th Mar. 2019)

11. Hoepman, J.: Privacy Design Strategies. In: SEC. IFIP Advances in Information and Communication Technology, pp. 446–459. Springer (2014)
12. Kamat, P. et al.: Temporal Privacy in Wireless Sensor Networks. In: 27th IEEE International Conference on Distributed Computing Systems (ICDCS 2007), June 25-29, 2007, Toronto, Ontario, Canada, p. 23. IEEE Computer Society (2007)
13. Karapiperis, D., Gkoulalas-Divanis, A., and Verykios, V.S.: FEDERAL: A Framework for Distance-Aware Privacy-Preserving Record Linkage. IEEE Transactions on Knowledge and Data Engineering 30(2), 292–304 (2018)
14. Kargl, F. et al.: privacypatterns.eu, (2019). `https://privacypatterns.eu` (visited on 29th Mar. 2019)
15. Kerschbaum, F.: 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, pp. 68–71. ACM (2007)
16. Lenhard, J., Fritsch, L., and Herold, S.: 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, pp. 194–201. IEEE Computer Society (2017)
17. Mattermost, Inc: Mattermost Server v4.8.0, (2018). `https://github.com/mattermost/mattermost-server/releases/tag/v4.8.0`
18. Mattermost, Inc: Mattermost Webapp v5.5.1, (2018). `https://github.com/mattermost/mattermost-webapp/releases/tag/v5.5.1`
19. Mattermost, Inc: Mattermost Website, `https://www.mattermost.org` (visited on 30th Mar. 2019)
20. McCulley, S., and Roussev, V.: 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, pp. 66–76. ACM (2018)
21. Microsoft: Workplace Analytics, `https://products.office.com/en-us/business/workplace-analytics` (visited on 30th Mar. 2019)
22. Ogriseg, C.: GDPR and Personal Data Protection in the Employment Context. Labour & Law Issues 3(2), 1–24 (2017)
23. Onoue, S., Hata, H., and Matsumoto, K.: A Study of the Characteristics of Developers' Activities in GitHub. In: 2013 20th Asia-Pacific Software Engineering Conference (APSEC), pp. 7–12 (2013)
24. Pandurangan, V.: On Taxis and Rainbows. Lessons from NYC's improperly anonymized taxi logs, (2014). `https://tech.vijayp.ca/of-taxis-and-rainbows-f6bc289679a1` (visited on 30th Mar. 2019)
25. Paverd, A., Martin, A., and Brown, I.: Modelling and automatically analysing privacy properties for honest-but-curious adversaries. Tech. rep., (2014)
26. Rastogi, A., and Nagappan, N.: 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, pp. 77–86. IEEE Computer Society (2016)
27. Roig, A.: Safeguards for the right not to be subject to a decision based solely on automated processing (Article 22 GDPR). European Journal of Law and Technology 8(3) (2017)
28. Slagell, A.J., Lakkaraju, K., and Luo, K.: FLAIM: A Multi-level Anonymization Framework for Computer and Network Logs. In: Proceedings of the 20th Confer-

ence on Systems Administration (LISA 2006), Washington, DC, USA, December 3-8, 2006, pp. 63–77. USENIX (2006)
29. Sweeney, L.: k-Anonymity: A Model for Protecting Privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10(5), 557–570 (2002)
30. Tursunbayeva, A., Lauro, S.D., and Pagliari, C.: People analytics - A scoping review of conceptual boundaries and value propositions. Int J. Information Management 43, 224–247 (2018)
31. Wang, N., and Katsamakas, E.: A Network Data Science Approach to People Analytics. Information Resources Management Journal 32(2), 28–51 (2019)
32. Wernke, M. et al.: A classification of location privacy attacks and approaches. Personal and Ubiquitous Computing 18(1), 163–175 (2014)
33. Yang, X. et al.: A novel temporal perturbation based privacy-preserving scheme for real-time monitoring systems. Computer Networks 88, 72–88 (2015)
34. Zhang, J., Borisov, N., and Yurcik, W.: 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, pp. 1–9. IEEE (2006)