Forensic Science International: Digital Investigation 49 (2024) 301755

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS USA 2024 - Selected Papers from the 24th Annual Digital Forensics Research Conference USA

Beyond timestamps: Integrating implicit timing information into digital forensic timelines

Lisa Marie Dreier [a,*], Céline Vanini [b], Christopher J. Hargreaves [c], Frank Breitinger [b], Felix Freiling [a,**]

[a] Department of Computer Science, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Erlangen, Germany
[b] School of Criminal Justice, University of Lausanne, 1015, Lausanne, Switzerland
[c] Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford, OX1 3QD, United Kingdom

ARTICLE INFO

Keywords:
Timelining
Relative ordering
Implicit timing information
Logical clocks

ABSTRACT

Generating timelines, i.e., sorting events by their respective timestamps, is an essential technique commonly used in digital forensic investigations. But timestamps are not the only source of timing information. For example, sequence numbers embedded in databases or positional information, such as the line numbers in log files, often contain implicit information about the order of events without directly referencing a timestamp. We present a method that can integrate such timing information into digital forensic timelines by separating sources of timing information into distinct time domains, each with its own timeline, and then connecting these timelines based on relations observed within digital evidence. The classical "flat" timeline is thereby extended into a "rich" partial order, which we call *hyper timeline*. Our technique allows ordering of events without timestamps and opens a rich set of possibilities to identify and characterize timestamp inconsistencies, e.g., those that arise from timestamp tampering.

1. Introduction

Generating timelines (an activity often called *timelining*) is an essential technique commonly used in digital forensic investigations. When sorting events by their respective timestamp, the approach heavily depends on the existence of such timestamps within digital data used as evidence. The technique is rather fragile since different timestamps might come from different sources or have different granularity so they are in general hard to compare. Their integration into a classical (flat) timeline therefore involves considerable thought and effort (Metz, 2021). Given sufficient effort, timestamps can — at least in principle — be aligned across different timezones, clock granularities and clock skews.

However, timestamps are not the only source of timing information embedded in digital evidence. For example, positional information in file systems (such as the position of file metadata in the MFT of NTFS) or within files (such as the line numbers of entries in the syslog log file) is often related to the order in which its content was created. Moreover, sequence numbers convey time-related ordering information, e.g., the primary keys in SQLite databases are generated incrementally by default so they can potentially be used to order table entries regarding their creation time. In contrast to *explicit* timing information (as expressed by classical timestamps) that is based on some idealized notion of metrical time, *implicit* timing information is based on any information not expressed as a classical timestamp.

1.1. Implicit Timing Information and its Relevance

The possibility to extend the information provided by timestamps by taking implicit timing information into account has been observed in digital forensic analysis, for example by Gladyshev and Patel (2005) and Marrington et al., 2011, but its connection to the creation of timelines is largely unclear (see also the discussion of other related work in Section 2). Since implicit timing information does not necessarily contain any

* Corresponding author.
** Corresponding author.
E-mail addresses: lisa.dreier@fau.de (L.M. Dreier), celine.vanini@unil.ch (C. Vanini), christopher.hargreaves@cs.ox.ac.uk (C.J. Hargreaves), frank.breitinger@unil.ch (F. Breitinger), felix.freiling@fau.de (F. Freiling).
URL: https://FBreitinger.de (F. Freiling).

https://doi.org/10.1016/j.fsidi.2024.301755

Available online 5 July 2024
2666-2817/© 2024 The Author(s). Published by Elsevier Ltd on behalf of DFRWS. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

information about how much time has passed between two events or at which time an event occurred, it can be regarded as a new class of timing information that is hard to integrate into classical timelines. It is therefore not surprising that information derived from such implicit timestamps is absent from the timelines produced by current tools.

Implicit timing information is also relevant for another reason: If perpetrators modify or delete timestamps to cover up illicit actions they might not be aware of implicit timing information. Since implicit timing information was often not created to keep track of time, its relation to event ordering has to be understood before it can be reliably interpreted. If it is overlooked by perpetrators, such information can therefore be used to reveal timestamp tampering.

1.2. Contributions

This paper presents a method for creating timelines that can handle both implicit timing information and conventional (explicit) timestamp data. The method is based on formal concepts that are implemented using a logical database able to infer further timing information from data inserted by parsers for explicit and implicit timing information.

The central idea of our method is to separate different types of timing information into distinct *time domains*. In general, a time domain is an environment in which timing information of a specific type can be compared. While the concept of time domain might appear abstract, it is motivated by real-world needs as different time domains allow to separate the concerns of different classes of timestamps, e.g., whether they represent implicit or explicit timing information, the granularity of a timestamp, the timezone used, or the interpretation details of a certain form of implicit timing information. Each time domain defines a separate "later-than" relation that is used to order events within that specific time domain, generating an independent timeline per time domain.

To connect these independent timelines, we observe that certain events have time information from different time domains. For example, website visits in a Firefox History contain a sequence number of the entry and a timestamp. Such events can be used to define special relations that link events from different time domains, thus creating an overall partial order of events which we call a *hyper timeline*. A hyper timeline integrates multiple *intra*-time domain relations to a single *inter*-time domain relation. Using a hyper timeline, global properties like "event a happened after event b" or "event a happened between event b and c" can now be automatically inferred by a logical reasoning mechanism incorporating all timing information extracted from the evidence.

Our approach has the potential to leverage any form of timing information for event reconstruction, be it explicit timestamps from different clocks of different granularity or implicit timing information like sequence numbers and positional information. As the latter kind of data was mostly not created to explicitly keep track of time and often is merely an inevitable consequence of an implementation, it is more robust than explicit timing information and therefore harder to tamper with such data, increasing the reliability of this type of source. So our method can also be used to detect inconsistencies in the data, creating opportunities for timestamp tamper detection.

To summarize, the contributions of this paper are:

- Based on a formal model, we present a general method to integrate arbitrary sources of time-related information into a single (partially ordered) digital forensic *hyper timeline*.
- Using several case studies, we argue that this method can be used to complement traditional timeline analysis.
- We provide a tool that can integrate this information together with timestamp-based information.

1.3. Outline

The paper is structured as follows: We begin by revisiting core concepts and methods of existing timeline approaches and thereby discussing related work in Section 2. In Section 3 we introduce and define the concept of hyper timelines. Its usefulness is then demonstrated in Section 4. Section 5 elaborates on tool support for hyper timelines. Section 6 discusses the limitations of the work, while Section 7 provides conclusions.

2. A Brief Timeline of Digital Forensic Timelines

We now present related work that used explicit and implicit timing information in digital forensic science.

2.1. Ideal Time

The common (and naïve, see below) understanding of time that also underlies the concept of classical timelining in digital forensic science is that of "standard time" (van Benthem, 1983), which we will call *ideal time* to prevent misunderstandings with standard time zones. Ideal time is understood to hold everywhere. Similarly, the passing of ideal time is usually assumed to be the same everywhere. Furthermore, ideal time is usually considered to be metric, i.e., measurable in metric time intervals (like seconds).

2.2. Clocks and Timestamps

A *clock* is a mechanism that measures time and gives it a representation. The clock reading is often called a *timestamp*. A common assumption is that there is a standard representation for ideal time, which we call an *ideal clock*. This typically is a reference to Coordinated Universal Time (UTC), the standard by which the world regulates clocks and time. An example timestamp in UTC expressed in ISO 8601 format is: *2024-01-04T18:52:28+00:00*, and includes a record of the timezone.

An ideal clock measures the passing of ideal time and can be used to assign (ideal) timestamps to events. Real-time clocks that have millisecond precision and are synchronized with official sources of time (like those given by atomic clocks) support the idealized view that the passing of time is the same and comparable everywhere and that the readings of such clocks can always be totally ordered.

2.3. Logical Clocks

Clocks commonly are seen as a mechanism to measure time in metric intervals like seconds. While modern technology gives the illusion of ideal time, the time-space continuum may not be standard (Minkowski, 1909), i.e., time may not be the same everywhere. But also for various other reasons, there may be bounds on the precision that globally synchronized clocks can have. For example, given the practical difficulty of clock synchronization especially in globally distributed systems, methods have been developed that capture the passing of time in a relativistic way. In this view, any clock that respects the succession of events can be used to represent time.

This was observed by Lamport (1978) who introduced the notion of *logical clocks*. Using a logical clock, a timestamp is a sequence number that can be used to order events according to potential causality: if event a may have influenced event b, then the timestamp of a must be less than the timestamp of b. Logical clocks therefore give indications of event orderings that Lamport denoted as the global *happened-before* relation. This shows that clocks and timestamps must not necessarily use the metric system to measure time.

2.4. Existing Timeline Approaches

Early timeline approaches have included extraction of timestamps from file systems, for example in The Coroner's Toolkit (Farmer and Venema, 2004) or The Sleuth Kit (Carrier), followed by more recent work that integrates timestamps from *within* files such as OS files, SQLite databases, and log files, rather than using only file system times. Notable

examples include the CyberForensic TimeLab (Olsson and Boldt, 2009) and Zeitline (Buchholz and Falk, 2005).

This approach of extracting as many timestamps as possible and integrating them into a single timeline has been coined as a "super timeline" based on (Guðjónsson, 2010), and is core to the mainstream digital forensic timeline tool log2timeline/plaso, and is an approach used in many forensic tools, including Autopsy (Carrier, 2021), Axiom (Magnet Forensics), and Cellebrite Physical Analyzer (Mahalik, 2021). While this is a powerful, well-developed, and widely accepted technique, there are many challenges to this approach.

The extraction of timestamps from a large number of digital forensic artefacts is challenging since correct parsing is required for each file format. This complexity is generally handled using software engineering approaches, e.g., use of plugin-based architectures (Plaso documentation), allowing new formats to be integrated by the community once a new timestamp source is identified.

The normalisation of timestamps during the integration into a super timeline is also a challenge (Metz, 2021), particularly with regard to timezones. More specifically, some times may be recorded as UTC (e.g., the MFT and many Windows OS timestamps) while others use local time (e.g., `setupapi.dev.log`). This challenge is also addressed programmatically, i.e., based on the specific plugin that is parsing the specific source but relies on the timezone being known.

There are also challenges associated with the analysis of these timelines since the volume of timestamps extracted is substantial (Chabot et al., 2014). Approaches have included heuristic-based approaches to extract "high-level" events (e.g., the connection of a USB stick) (Hargreaves and Patterson, 2012; Bhandari and Jusas, 2020a, 2020b), as well as the application of colour coding rules to highlight specific entries associated with these events (Lee, 2012; Debinski et al., 2019). More recent approaches use machine or deep learning techniques to detect certain events of interest in timelines (Studiawan et al., 2020; Studiawan and Sohel, 2021; Dusane and Sujatha, 2022), visualize artefacts relationships (Henseler and Hyde, 2019), or rank artefacts according to their context-based relevance (Du et al., 2020).

Regarding the source timestamps, on which any analysis is based, locally stored timestamps are only as accurate as the system clock at the point in time at which that timestamp was retrieved. This is mostly handled with a broad approach of applying a "skew" to the timeline, usually based on any difference between the system clock and a trusted time source at the time of acquisition. Tools allow the application of this skew to the entire timeline (e.g., using a `–skew` parameter in plaso). However, this approach does not consider previous points in time where the system clock may have been incorrect but has been subsequently corrected, either through natural clock drift or through system clock tampering. There is some work describing specific approaches for incorrect clock detection (Weil, 2002; Schatz et al., 2006; Kaart and Laraghy), but no generalised approach is currently available in research or tools.

It is also possible for timelines to be generated and visualized using multiple sources of digital evidence (Patterson and Hargreaves, 2012; Berggren, 2018). This requires normalisation of time skew on multiple sources of digital evidence, assuming clock skew is known for them all.

2.5. Timestamp Granularity

Different timestamps often have different granularity, meaning that different timestamp fields may have different resolutions. This has two implications. First, when comparing timestamps of different resolutions, they have to be adjusted to the lower granularity (e.g., remove nanoseconds). Second, the lower granularity is less precise which means two events could be different in a higher resolution but seem identical in a lower resolution.

The precise order in which to place these events in a super timeline is undefined. Hargreaves and Patterson (2012) handle this issue using intervals, i.e., with each "low-level" (extracted) timestamp having a *min* and a *max* field, allowing the uncertainty of extracted lower resolution timestamps to be captured. Such approaches, however, create additional complexity that is hard to visualize and handle, so it is still absent from the mainstream tools.

2.6. Using non-timestamp based information

The focus on explicit timestamps appears to be a trait specific to digital forensic science, presumably because explicit timestamps are in excessive supply. It is well-known from other scientific areas that the ordering of events can be inferred using implicit timing information, i.e., without having explicit timestamps. Some examples are the study of sedimentation for dating purposes in archaeological stratigraphy (Harris, 1989) or ordering a complete set of grave goods based on the statistical occurrence of temporary fashion trends (Renfrew and Bahn, 1991). While these fields appear detached from digital forensic science, similar observations can be made for digital data. For example, sequence numbers associated with events represent logical ordering information similar to the statistical occurrence of fashion trends in a set of grave goods, and positional information such as line numbers in logfiles contain positional ordering information similar to stratigraphy (Casey, 2018).

There has been some work attempting to extract logical relations from digital evidence to aid event reconstruction. For example, Gladyshev and Patel (2005) (later extended by James and Gladyshev (2015)) propose the method of event time bounding by inferring the execution time of an event (without explicit timestamp) from events with timestamps that happened before and after that event. Based on Lamport's *happened-before* relation, Willassen (2008a) and later Levett et al., 2010 generalize this approach by observing that causal dependencies in data can be used to check the validity of assumptions, e.g., those on the accuracy of clocks (see also some refinements by Willassen (2008e,c,d,b, 2009)). Based on these ideas, Marrington et al., 2011 provided tool support for detecting such inconsistencies in digital forensic timelines. The tool can, for example, detect that logon/logoff events are in the wrong order or have been removed.

Overall, these ideas have been frequently re-invented in the literature, mostly in the context of file systems (Lee et al., 2019; Bahjat and Jones, 2019; Tse, 2011; Li et al., 2016). They are mainly concerned with consistency checking of timelines using hard-coded self-implemented rules rather than integrating such information in digital forensic timelines.

2.7. Summary

Timelines are an important analysis approach in digital forensics, and both research and practical implementations have resulted in an approach of extraction, normalisation, integration, and ordering of timestamps into a single timeline representing a digital device's history. Many of the challenges that result from this approach have been considered by the research community, but they commonly fall into one of two categories: (1) the practical and tool-oriented approaches based on a common "flat" timeline, and (2) the more formal and largely impractical approaches that extend flat timelines with additional information derived from the input. In this work, we propose a novel approach that reconciles both approaches, yielding a generic method to integrate implicit timing information into practical digital forensic timelines supported by tools.

3. Integrating Implicit Timing Information into Digital Forensic Timelines

While the time domain of ideal time is, as mentioned above, a reference to UTC, not all timestamps can be readily compared. For example, the readings of different (unsynchronized) clocks cannot be compared if the clock skew is unknown. A comparison to the reading of a

logical clock appears similarly futile without a common reference point. We resolve this issue by separating these different clock readings into different time domains.

3.1. Time Domains and Time Domain Entries

A *time domain* D is a set of time values paired together with a context in which they can be compared. The values from D are those extracted from some piece(s) of evidence.

Example 1. For example, all the 'file creation' timestamps within an NTFS file system are stored as 64-bit values representing the number of one hundred nanoseconds since January 1, 1601 UTC (*Carrier, 2005*). On a single computer, they are assumed to be comparable and therefore belong to one time domain D_1.

Example 2. The lines of the Linux logfile syslog begin with a timestamp expressed in local time. Therefore, without knowing the timezone, they cannot be directly compared to the UTC timestamps in D_1, so we assign a different time domain to them which we denote D_2. Aside from the timezone difference, the contrasting time resolution (recorded to the second rather than one hundred nanoseconds) would cause an assignment to a different time domain.

Example 3. In SQLite databases, the primary key in a table (stored in the column `ID`) is a 64-bit signed integer that uniquely identifies the row in the table. When a new value is inserted, by default the new primary key is one more than the largest primary key currently in use (*Kreibich, 2010*). Primary keys therefore resemble the readings of a logical clock and can be collected in a (per table) time domain D_3.

Example 4. Even though each line in *syslog* begins with a local timestamp, the line number of every entry implicitly carries information about the order in which the entry was added to the file. For each such log file, these numbers are comparable in their time domain D_4 but incomparable with entries in the other time domains. Note that while it may appear that time domain D_2 expresses the same order as time domain D_4, the entries in both time domains are created through different mechanisms. Separating these time domains opens the possibility to check that D_2 and D_4 in fact contain the same order.

Since the value v from a time domain does not make sense in isolation, we call the pair (v, D) a *time domain entry*. In practice, D may merely be a reference to a specific time domain from a set of identified domains in the data.

For each time domain D, we assume an intra-time domain ordering relation $<_D$ exists for the values of D. Hence, given two time domain entries (v, D) and (v', D), we can order them by comparing v and v' using $<_D$. To be useful, the ordering relation must at least be transitive, i.e., if $a <_D b$ and $b <_D c$, then it must hold that also $a <_D c$.

Example 5. Some time domain entries for Examples 1 to 4 above are:

1. (133,485,408,000,000,000, D_1) refers to the number of one hundred nanoseconds since January 1st, 1601 UTC (Carrier, 2005) and D_1 refers to an NTFS file system on a disk volume.
2. (*Jan 28 15:51:50*, D_2) is a time domain entry in the file `/var/log/syslog` on a computer.
3. (15, D_3) is a time domain entry in a table of an SQLite file on a computer.
4. (3577, D_4) is a time domain entry for line number 3577 in the file `/var/log/syslog` on a computer.

While the ordering relations $<_{D_1}$, $<_{D_2}$, $<_{D_3}$ and $<_{D_4}$ within the time domains D_1 to D_4 in the above examples are obvious, i.e., 1st Jan 2024 is less than 2nd Jan 2024, and row 42 is less than row 100, in contrast, entries from different time domains like (Jan 28 15:51:50, D_2) and (3577, D_4) cannot be compared, at least not directly.

3.2. Connecting Time Domains

Using multiple time domains prevents incorrect comparisons between timestamps. However, it introduces the necessity to make seemingly incomparable time domain entries comparable. This can be done by defining relations for the time domain entries extracted from the evidence.

Let $\mathbb{D}$ be the set of all time domains and $\mathbb{V}$ be the set of all possible values that any time domain $D \in \mathbb{D}$ can have. Then $\mathbb{V} \times \mathbb{D}$ describes the set of all possible time domain entries. Using the local precedence relations and additional observations, we want to derive a *global ordering relation* $<$ and a *global equality relation* $=$ on time domain entries. Formally, $<$ and $=$ are subsets of $(\mathbb{V} \times \mathbb{D}) \times (\mathbb{V} \times \mathbb{D})$.

Let (v, D) and (v', D') be two time domain entries from different time domains. There are now two possible relations that can be observed in the evidence:

1. Coincidence: (v, D) and (v', D') refer to the *same point in time*. This happens when the two time domain entries were created at the same point in time (e.g., as part of an assumed atomic transaction) or were intended to express the "same time".
2. Precendence: (v, D) is witnessed by (v', D'), i.e., (v, D) existed *before* (v', D') was created.

Intuitively, Precedence is a very similar concept to Lamport's *happened-before* relation (Lamport, 1978) only that it is derived from the context of the evidence instead of specially constructed timestamps. In a sense, Precedence is also more generic than Coincidence because Coincidence can be regarded as mutual Precedence, i.e., where (v, D) witnesses (v', D') and vice versa.

We give some examples and thereby argue that Precedence and Coincidence can be observed almost ubiquitously in digital data.

Example 6. **(Coincidence)** Syslog is used in many Linux systems to record general system information into the file `/var/log/syslog`. This is a text file in which each log message is separated by a line separator. The entry format per line consists of a timestamp (month, day and local time) with a precision of one second. Fig. 1 shows a sample on a computer running Ubuntu Desktop 20.04.

The explicit timestamps at the beginning of each line define entries in time domain D_2 whereas the line numbers[1] of that file are entries in time domain D_4. Since the line number implicitly is "created" when the timestamp is written to the file, the timestamp is in Coincidence with the line number. For example, in the file depicted in Fig. 1, the time domain entry (348, D_4) is in Coincidence with time domain entry (Jan 28 15:51:52, D_2) and (350, D_4) is in Coincidence to (Jan 28 15:51:53, D_2).

Precedence occurs regularly if two time domain entries are associated with the send and receipt of messages. In contrast to the use of logical clocks, Precedence here spans different time domains.

Example 7. **(Precedence)** The Firefox history and the Firefox cache are used to temporarily store data about visited websites locally for an improved usage experience of the browser. The Firefox history is stored in an SQLite file called `places.sqlite` in the user's Firefox profile, whereas the Firefox cache (version 2) stores one cache file for each cached website in the `cache2/entries` (Metz, 2018). Within each cache file, a copy of the HTTP response header returned by the server is stored, including the timestamp when the server generated its response in an RFC 7231 compliant representation. The Firefox history contains a column `visit_date` supposedly recording the time that a specific website was visited (as a Unix timestamp taken from the local clock of the computer). While it may be expected that the (local) timestamp in

[1] The original file does not have line numbers. We added them here for illustration purposes.

```
347  Jan 28 15:51:52 asterix gnome-shell[3363]: libEGL warning: egl: failed to create dri2 screen
348  Jan 28 15:51:52 asterix gnome-shell[3363]: Failed to initialize glamor, falling back to sw
349  Jan 28 15:51:53 asterix systemd[817]: Reached target GNOME session X11 services.
350  Jan 28 15:51:53 asterix systemd[817]: Starting GNOME XSettings service...
```

Fig. 1. Excerpt from `/var/log/syslog` from computer `asterix`. Each line has been prefixed with line number in the file for illustration.

the history and server timestamp refer to the same point in time, it has been observed that entries to the history should not be made before the request is issued to the server (Groß et al., 2020), so overall we observe Precedence, i.e., the visit date in the history is a witness of the server timestamp.

We assume that observations of Coincidence and Precedence have been extracted from a set of digital evidence according to a set of generally accepted (or at least documented) rules. We can now define the two global relations $<$ and $=$ on the set of all possible time domain entries as follows:

- $(v, D) < (v', D')$ if and only if $v <_D v'$ in case $D = D'$ or (v, D) is observed to be in Predence of (v', D') in case $D \neq D'$.
- $(v, D) = (v', D')$ if and only if there is an observed Coincidence of (v, D) and (v', D').
- The union of $<$ and $=$ is transitive, e.g., if $(v, D) < (v', D')$ and $(v', D') = (v'', D'')$ then $(v, D) < (v'', D'')$.
- $=$ is also reflexive, i.e., for all (v, D) holds that $(v, D) = (v, D)$.

Note the subtle difference between the definitions of $<$ and $=$: While the local precedence relation $<_D$ carries over to the global precedence relation $<$, the local equality relation does not extend globally. Local equality does not imply global equality, as exemplified by the case study of poor granularity explained in Fig. 2.

3.3. From Super Timelines to Hyper Timelines

Overall we assume that some amount of digital evidence has been acquired and that investigators have defined a set of time domains such that the evidence contains numerous time domain entries from different time domains. Assume we have extracted all time domain entries that are contained in the evidence. We call the resulting set of time domain entries the *evidence set*.

While the approach of a *super timeline* (Guðjónsson, 2010) integrates all time domain entries from the evidence set into a normalized yet "flat" linear sequence, we attempt to infer a global ordering of timestamps that respects individual time domains. As a result, we determine the global relations $<$ and $=$ that allow to compare timestamps across time domains while respecting their individual (local) precedence relations per time domain. In a sense, it creates a higher-order timeline which—in accordance with higher order properties in distributed systems (*hyperproperties*) (Clarkson and Schneider, 2008)—we call a *hyper timeline*.

In practice, it is essential to maintain *provenance information* of observed relations. In classical timelines, tools like Plaso provide information about the source of a specific timestamp in the timeline. For hyper timelines, tools should also maintain such provenance information not only for time domain entries directly extracted from evidence but also for the Coincidence and Precedence relations that are extracted from it (e.g., which file or database table does the information come from). While we have omitted such aspects from the formalization above and from the tool to handle hyper timelines discussed below, both can in principle (and will) be extended to handle additional provenance information.

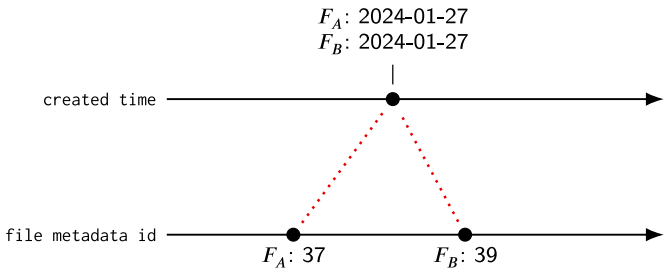

Fig. 2. Reasoning across time domains can disambiguate timestamps of poor granularity.

4. Case Studies

This section provides a series of examples that illustrate when applying the concept of time domains is beneficial in digital forensics.

4.1. Ordering Events with Limited Granularity

Reasoning across time domains can be helpful if timestamps have insufficient granularity. A particularly instructive case for timestamps of different resolutions is the file system FAT, where the resolution of created time is 10 ms, while write time has a resolution of 2 s and access time has a resolution of 1 day (Carrier, 2005).

Imagine some file system had a created timestamp granularity which is precise only to the day. If two files, F_A and F_B, are created on the same day, timestamps cannot answer the question of which file was created first. If the order in which files are created is also expressed in a different time domain, this can be used to disambiguate the poor granularity of the original time domain. So if file system metadata entries are created incrementally (like MFT entries in NTFS or inode numbers in Ext), the creation order of files F_A and F_B can in fact be derived (see Fig. 2).

4.2. Finding Indications of Tampering

Timestamp tampering comes in many different forms. One option is to manually turn off time synchronization and then set the hardware clock to a certain time. Local timestamps like those in file system metadata will then be taken from the modified time source. Such tampering is notoriously difficult to detect when only considering one time domain. Given Coincidence or Precedence relations to other time domains, such tampering can be more easily detected, because timestamp readings may contradict corresponding entries in other time domains.

As an example, consider the data given in Table 1. It is an excerpt from the Firefox browser history in an educational case where a suspect was accused of attempted homicide on the evening of May 26th, 2016. The suspect had claimed that she had watched Youtube videos during that time, thus creating an alibi. Superficial analysis of the Firefox browser history on the suspect's computer appears to indicate that there had been browser activity during that time. However, taking the primary keys of the browser history entries as second time domain and presuming Coincidence between the two time domain entries, an inconsistency arises.

To see this, consider Fig. 3 where the data of Table 1 is visualized as a hyper timeline. The upper time domain represents the sequence of

Table 1
Circular precedence relation after adjusting the system clock.

`ID` in `moz_places`	`visit_date` in `moz_places`
338	2016-05-26 19:49:05
373	2016-05-26 19:47:26
378	2016-05-26 19:47:39

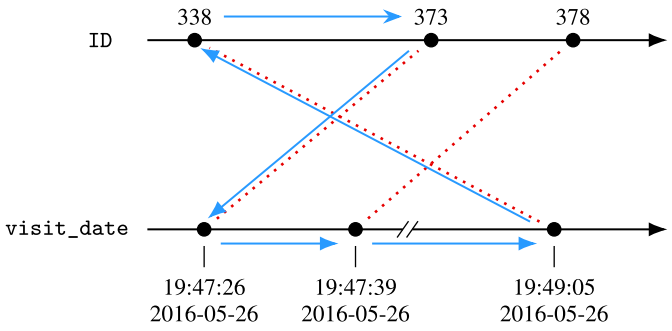

Fig. 3. Inconsistency arising from changing the local clock: circular precedence relation (expressed as arrows) indicates that either the order in the time domain `ID` or in the time domain `visit_date` of the Firefox history must be wrong. The dotted lines represent Coincidence relation.

primary keys while the lower represents the sequence of visit timestamps. The Coincidence relations extracted from the table are indicated as dotted-lined (equality) connections between time domain entries. We can now see that the global precedence relation has a cycle (looking like an hourglass, indicated by arrows): The value 338 precedes 373 in the upper time domain, and the three values in the lower time domain also precede each other. The Coincidence relations can be used to "jump" from upper to lower time domain and back, resulting in the cycle. In the end, value 338, for example, precedes itself, which cannot be true.

4.3. Determine Timezone without Explicit Value

Obtaining the timezone for a machine is performed early on in the analysis process so that the correct timezone can be applied to the many UTC timestamps recorded on a system. On Windows for example, this can be obtained from the Windows Registry (Wilson, 2010), however, this is the last recorded timezone for the machine before it was powered off.

Time domains provide a convenient mechanism to determine the timezone of the machine at other points in time. For example, if we consider the connection of a USB mass storage device. This makes multiple changes to a Windows system, including `setupapi.dev.log`, Windows Event logs, and the Windows Registry (Carvey and Altheide, 2005; Deb and Chetry, 2015; Arshad et al., 2018).

Since entries in the Windows Event Logs and Last Written times for keys in the Windows Registry are stored as UTC, and times in `setupapi.dev.log` are in local time, these would be stored in different time domains. With knowledge applied from digital forensic artefact research for the first connection time of a USB device, a Coincidence can be derived for these events across time domains, i.e., they can be considered to have happened at the same point in time (see Fig. 4). The difference in the absolute values recorded across these different time domains allows the calculation of the timezone of the machine at the time of this event, rather than the final timezone setting of the machine

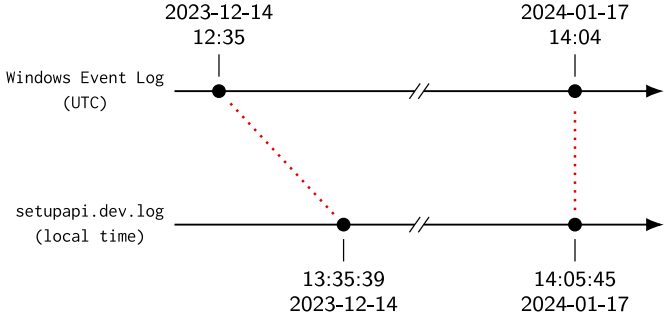

Fig. 4. Coincidence between time domain entries in local time (below) and UTC (above) allow continuous automated timezone estimation.

before imaging.

5. Tool Support for Hyper Timelines

Formally, a hyper timeline can be regarded as a graph where nodes are time domain entries and edges are the relations between them. Some relations are directly extracted from evidence while others are inferred using a logical reasoning mechanism. A tool that supports the handling of hyper timelines must therefore provide a database for time domain entries and relations between them. Additionally, it must provide a reasoning mechanism that is used to make inferences. Similar to established tools like Plaso, we also need a library of modules that extract specific time domain entries and relations from evidence and ingest them into the database. The complete architecture is depicted in Fig. 5.

Based on it, we have implemented a prototype that demonstrates the feasibility of our approach. We will now discuss the individual modules of the prototype using the case study in Section 4.2 as running example. There, a suspect had claimed to have watched videos on YouTube at the time of crime and the task in this educational case was to investigate whether this statement can be supported by the digital evidence.

5.1. Handling Input Variety

Above we have argued that our formalization can be utilized for a variety of purposes, and thus, can process a wide range of data. As we do not want to limit this range by selecting a certain parser for one kind of data, we take a more general approach: For each example case, we used an appropriate forensic tool, e.g., X-Ways Forensics or DB Browser for SQLite, to identify different time domains and to extract data for each of them into simple spreadsheet files (tab-separated-values, `.tsv`). This file format is generic enough to represent a large variety of data, while being supported by many tools. Thus we decided to use it as an input file format for the ingest module.

In our example case, we used DB Browser for SQLite to extract data from the Firefox History file `places.sqlite` for two relevant time domains: (1) the timestamp of the entry and (2) the sequential identifier of the entry. Similar to handling classical super timelines, it is possible to restrict data extraction, e.g., to a particular time frame of interest. In our case, the time of the crime is between 7 p.m. and 8 p.m. on May 26, 2016. So we created an appropriate query in DB Browser and extracted the resulting view into a `.tsv` file, an excerpt of which can be seen in Fig. 6.

5.2. Ingest Module

The `.tsv` files are used as input for a Python script, which extracts the central entities of our formalization scheme. It interprets columns as potential time domains, their header lines as time domain references and all other column entries as possible time domain entries. The selection of time domains must be performed by the analyst by passing the column numbers and data types of the time domains together with the relations (Coincidence or Precedence) between them to the script. Based on this information, the script parses the `.tsv` file, extracts time domains as well as time domain entries and inserts them together with their corresponding relations into a database.

This process is illustrated in our example data in Fig. 6: from the context of the case, one is able to specify that columns two and three are time domains containing a long and a datetime value, respectively, with their time domain entries being connected by a Coincidence relation. Based on this information, the script is able to extract the time domains `visit_date` and `ID` together with 27 time domain entries for each of them. It inserts all of this into the database and adds one global equality relation between each pair of time domain entries corresponding to table entries on the same line of the `.tsv` file.

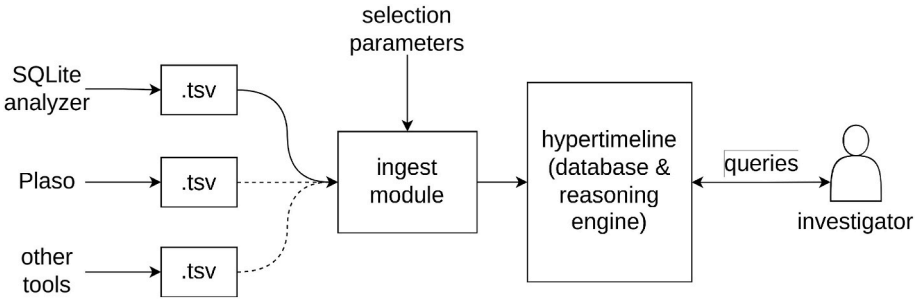

Fig. 5. Architecture of tool support for hyper timelines.

url	visit_date	ID
http://forum.gofeminin.de/forum/f252/__f193_f252-Lange-verheiratet-meine-Frau-geht...	2016-05-26 19:45:16	336
https://www.elitepartner.de/forum/frage/mein-partner-betruegt-mich-seit-monaten.64296/	2016-05-26 19:45:20	337
http://youtube.com/	2016-05-26 19:47:26	373
http://www.youtube.com/	2016-05-26 19:47:26	374
https://www.youtube.com/	2016-05-26 19:47:26	375
https://www.youtube.com/	2016-05-26 19:47:28	376
https://www.youtube.com/results?search_query=berserk+movie	2016-05-26 19:47:35	377
https://www.youtube.com/watch?v=1nYSC08rbMI	2016-05-26 19:47:39	378
https://www.youtube.com/watch?v=AMd2AbEEcho&list=PLlwLXQJ7gn5bzCN5QDejbN...	2016-05-26 19:49:05	338
https://www.youtube.com/watch?v=9ivu-SB7X_s&list=PLlwLXQJ7gn5bzCN5QDejbNm...	2016-05-26 19:49:11	339

Fig. 6. Excerpt of the data extracted from `places.sqlite` with two different time domains.

5.3. Database with Reasoning Engine

The core of our tool is a database: The schema is based on our formalization, written in a high-level abstract language for defining formal concepts as entities, attributes and relations. This allows the database not only to typecheck its content but also to infer new information based on its content and on rules given by the schema.

Out of the possible database candidates, we selected TypeDB (Vaticle Ltd., 2024b) because of its flexibility and personal familiarity. TypeDB is an experimental database with a conceptual data model, a strong subtyping system and a symbolic reasoning engine released under the GNU Affero General Public License (version 3). It offers a declarative query language called TypeQL, in which schemas can be declared and queries can be formulated. We note that any other database with an inference engine can be used.

Fig. 7 shows the TypeQL declarations of the concepts *time domain* and *time domain entry* as entities with attributes (`owns`) involved in relations (`plays`). While TypeDB offers powerful abstractions, it is still experimental, so we need to explain the peculiarities of the code: First, an attribute can be added arbitrarily often to an object, but the cardinality cannot be restricted in TypeQL. Hence, we cannot represent the fact that one time domain entry can only have one value. Second, an attribute needs to have a defined data type and cannot have a purely abstract data type defined only in the subtype. According to the developers (Vaticle Ltd., 2024a) this feature should be added in the next major release. For the time being, we decided to use one attribute per data type for the value of the time domain entry and add an additional relation between the time domain entry and its value to easily retrieve the corresponding value.

```
define time_domain sub entity,
    owns time_domain_reference,
    plays entry_of_domain:time_domain;

define time_domain_entry sub entity,
    owns time_domain_reference,
    owns value_long,
    owns value_datetime,
    plays precedence:earlier,
    plays precedence:later,
    plays global_ordering:earlier,
    plays global_ordering:later,
    plays global_equality:left,
    plays global_equality:right,
    plays entry_of_domain:entry,
    plays time_data_in_entry:entry;
```

Fig. 7. Declarations of time domain and time domain entries as TypeDB schemas.

As a third peculiarity, the connection between time domain and time domain entry is noteworthy: They are connected using a relation, but the entry also has an attribute containing the time domain reference. This is because entities are defined by their attributes, but not by their relations, and so omitting the extra attribute would turn two time domain entries with the same value but different interpretations into the same object.

After having defined the main entities, we also need to define the main relations: The global equality relation $=$, the global ordering relation $<$ and precedence are defined directly as relations. As the coincidence relation would exactly match the global equality relation $=$, we decided to skip defining the coincidence relation. Instead, the ingest module directly inserts the global equality relation $=$ and precedence. As by definition, the ordering relation $<$ is inferred by two inference rules: each precedence relation leads to a global ordering relation $<$ (as exemplified in Fig. 8) and each intra-domain-ordering as well. Additionally, properties such as transitivity are modelled as inference rules as well.

Applying this to our example case means that the database infers intra-domain ordering based on the values of the time domain entries: In the `visit_date` time domain, the time domain entry with value 337 is earlier than the one with value 338, which is transitively earlier than, e.g., the time domain entry with value 378. The same applies to the time domain `visit_date`: The time domain entry with value 2016-05-26 19:47:39 is earlier than the one with value 2016-05-26 19:49:05 and

```
rule global_ordering_if_precedence:
when {
    $te1 isa time_domain_entry;
    $te2 isa time_domain_entry;
    $lt12 (earlier:$te1, later:$te2) isa precedence;
} then {
    (earlier:$te1, later:$te2) isa global_ordering;
};
```

Fig. 8. Definition of how to infer global ordering relation $<$ from precedence relation in TypeQL.

so on.

Through the global equality relations between the time domain entry with value 338 in time domain `ID` and the one with value 2016-05-26 19:49:05 in `visit_date` and between the time domain entries with value 378 in time domain `ID` and the one with value 2016-05-26 19:47:39 in `visit_date`, the database will also infer further information: the time domain entry with value 2016-05-26 19:49:05 is later than the one with value 2016-05-26 19:47:39 and thus later than the one that happened at equal time: the one with value 378.

Furthermore, the time domain entry with value 378 is later than the one with value 338, which is equal in time to the one with value 2016-05-26 19:49:05. As a result, the time domain entry with value 378 is both, earlier and later than the one with value 2016-05-26 19:49:05, which can be regarded as an inconsistency. Note that inference rules only get applied as soon as a query requires them to be computed.

5.4. Query Interface

To use this formalization, an examiner has to query the database using the inference feature. For our work, we assume that an investigator mainly queries the database to show all global ordering relations between time domain entries *A* and *B*. Such a query can either be created directly by the investigator as formulated in Fig. 9 or hidden in a script to increase usability. For our case studies, the execution time in a Virtual Machine ranged from seconds up to 2 min for around 800 time domain entries, which is quite acceptable for a proof-of-concept implementation.

The answer is shown in Fig. 10 using the "Graph View" of TypeDB, but it could also be retrieved as machine-readable data, e.g., JSON or TypeQL. Looking at the result of the query, we can see one time domain entry that is both, earlier as well as later in time, compared to the other. This manifests the inconsistency mentioned above. Such inconsistencies should then be examined closely, either manually or by tracing the inference steps the database made using a special "explainable" feature of TypeDB. Doing so in our example, an investigator could find that the inconsistency was caused by a change of the local clock by the suspect, who wanted to antedate the video session to create an alibi for the time of the crime. Note that such inconsistencies (namely cycles in the precedence relation) can also be explicitly queried using TypeQL.

6. Discussion

While being more flexible than classical timeline, hyper timelines require careful consideration before being applied.

The first and foremost difficulty when creating a hyper timeline is the choice of time domains. This is a delicate issue since it considerably affects the possibilities of analysis. To increase or optimize the precision of analysis, one may opt for *many* time domains. For example, if the clock of a computer has been set back at some point in time, the time domain of all created timestamps of files in a file system may be split into multiple time domains to keep timestamps comparable within each fragment. Any observed inconsistency (detected as cycle in the global precedence relation, see Section 4) can be "fixed" by fragmenting time domains. This method can also be applied if the ordering relation in one time domain may be uncertain, e.g., due to a wraparound of sequence numbers or file system allocation positions when the end of the allocation area is reached.

However, having too many time domains causes fragmentation of the hyper timeline, resulting in sub-timelines that have no connection. Too few time domains make analysis simpler but also less precise in the sense that inconsistencies may remain unnoticed. This is exemplified by the fact that a classical timeline can be regarded as a hyper timeline with a minimum number of time domains, namely one.

Intertwined with the problem of choosing an appropriate number of time domains is the problem of correctly interpreting them. Even though explicit time domains are designed to express time, issues such as unspecified time zones or clock shifts in practice may hamper their interpretations. But with implicit timing information not being designed to express time, this issue severely aggravates: to be able to interpret implicit timing information, an investigator must thoroughly analyze the program or driver, which creates new time domain entries, to learn its behavior. Knowing how it handles the reuse of deleted entries and other details of the allocation strategy enables the investigator to determine suitable time domains and fragment them properly. In practice, this remains a highly intricate topic—especially for positional information—as driver behavior is not necessarily deterministic and may vary between different versions of a driver. Thus, further research in this area is needed.

Developing conclusive guidelines for the optimal choice of time domains is an open problem for future work. To make progress in this direction, hyper timeline construction and analysis tools should provide the possibility to dynamically select and deselect time domains from a library of well-chosen time domains during analysis. If an inconsistency vanishes after deselecting a particular time domain, a relation within that (deselected) time domain must be the cause of the inconsistency and can be investigated further.

Ideally, one would like to combine the power and precision of hyper timeline analysis with the simplicity and intuitiveness of classical timelines. And under certain conditions, it may be possible to create better super timelines using hyper timelines. For example, timestamps from clocks with different skews are seamlessly processed within a hyper timeline (i.e., without any alignment calculations) and can then be "flattened" into a super timeline respecting the global order of timestamps. To do this, we would need the observation of sufficiently many Coincidence relations between the time domains such that the partial order is reduced to a total order. Without a sufficient number of synchronization points, such automatic alignment will also be hard using hyper timelines. Similarly, implicit timing information also cannot be easily merged with classical timelines. It may, however, be possible to identify or choose one time domain and visualize implicit timing information as intervals which are spread over a classical timeline,

```
match
    $entry_A isa time_domain_entry, has time_domain_reference "ID", has value_long 378, has value_long $value_A;
    $entry_B isa time_domain_entry, has time_domain_reference "ID", has value_long 338, has value_long $value_B;
    $lt isa global_ordering;
   {$lt (earlier:$entry_A, later:$entry_B) isa global_ordering;} or {$lt (later:$entry_A, earlier:$entry_B) isa global_ordering;};
get $entry_A, $value_A, $entry_B, $value_B, $lt;
```

Fig. 9. Querying for all global ordering relations between two time domain entries *A* and *B* in TypeDB.

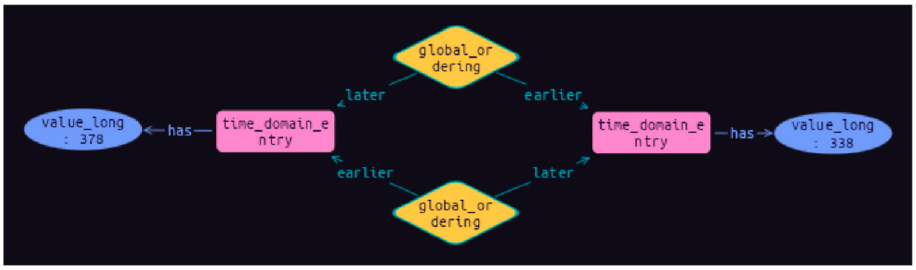

Fig. 10. Graph depicting the answer to the example query: Time domain entry *A* is both, earlier and later, than time domain entry *B*.

identifying the earliest and latest point in time of the classical timeline when an event might have occurred.

7. Conclusions

The article proposed the concept of hyper timelines. Hyper timelines are a flexible concept that allows the integration of implicit timing information into digital forensic timelines by separating sources into distinct time domains. Each time domain has its timeline, and these timelines are then connected based on relations observed within digital evidence. This concept enables the ordering of events without timestamps and thereby opens a rich set of possibilities to identify and characterize timestamp inconsistencies. Additionally, it allows to improve the precision of timestamp-based analyses, e.g., by ordering events with limited granularity or determining a timezone without an explicit value. But while the increased analysis flexibility of hyper timelines appears obvious, the usability of this concept in practice still needs to be explored. As first step in this direction, we wrote a proof-of-concept implementation and release it publicly.[2]

The practicality of hyper timeline analysis rests on the amount and quality of Coincidence and Precedence links that have been identified in data and can be extracted from evidence. While we have presented a couple of examples for these relations in this paper, it is left to future work to identify and "decipher" other such ordering relations and thereby lift the treasure trove of implicit timing information that is currently still hidden in digital evidence sets.

CRediT authorship contribution statement

Lisa Marie Dreier: Conceptualization, Investigation, Methodology, Software, Validation, Writing - Original Draft, Writing - Review and Editing. **Céline Vanini:** Conceptualization, Methodology, Writing - Original Draft, Writing - Review and Editing. **Christopher J. Hargreaves:** Conceptualization, Investigation, Methodology, Resources, Supervision, Writing - Original Draft, Writing - Review and Editing. **Frank Breitinger:** Conceptualization, Supervision, Writing - Review and Editing. **Felix Freiling:** Conceptualization, Methodology, Investigation, Writing - Original Draft, Writing - Review and Editing, Supervision.

Acknowledgments

Thanks to Florian Frank, Jan Gruber, Merlin Humml, Jenny Ottmann, Lutz Schröder, and Paul Wild for helpful discussions. Work was supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the Research and Training Group 2475 "Cybercrime and Forensic Computing" (grant number 393541319/GRK2475/2-2024).

[2] https://github.com/hypertimeline/hypertimeline.

References

Arshad, A., Iqbal, W., Abbas, H., 2018. Usb storage device forensics for windows 10. J. Forensic Sci. 63, 856–867.
Bahjat, A.A., Jones, J., 2019. In: Deleted File Fragment Dating by Analysis of Allocated Neighbors, vol. 28 S60–S67. doi: 10.1016/j.diin.2019.01.015. URL: https://www.sciencedirect.com/science/article/pii/S1742287619300258.
van Benthem, J.F.A.K., 1983. The Logic of Time. Springer.
Berggren, J., 2018. Timesketch. https://timesketch.org/.
Bhandari, S., Jusas, V., 2020a. An abstraction based approach for reconstruction of TimeLine in digital forensics. Symmetry 12, 104. https://doi.org/10.3390/sym12010104.
Bhandari, S., Jusas, V., 2020b. The phases based approach for regeneration of timeline in digital forensics. In: 2020 International Conference on INnovations in Intelligent SysTems and Applications (INISTA), pp. 1–6. https://doi.org/10.1109/INISTA49547.2020.9194649.
Buchholz, F.P., Falk, C., 2005. Design and implementation of zeitline: a forensic timeline editor. In: DFRWS.
Carrier, B., . Sleuthkitwiki - timelines. URL: http://wiki.sleuthkit.org/index.php?title=Timeline..
Carrier, B., 2005. File System Forensic Analysis. Addison-Wesley.
Carrier, B., 2021. Timeline Analysis. https://www.sleuthkit.org/autopsy/timeline.php, Accessed: 2024-01-24.
Carvey, H., Altheide, C., 2005. Tracking usb storage: analysis of windows artifacts generated by usb storage devices. Digit. Invest. 2, 94–100.
Casey, E., 2018. Digital Stratigraphy: Contextual Analysis of File System Traces in Forensic Science. Journal of forensic sciences 63, 1383–1391. https://doi.org/10.1111/1556-4029.13722.
Chabot, Y., Bertaux, A., Nicolle, C., Kechadi, M.T., 2014.A. Complete Formalized Knowledge Representation Model for Advanced Digital Forensics Timeline Analysis. In: Digital Investigation 11, S95–S105. https://doi.org/10.1016/j.diin.2014.05.009.
Clarkson, M.R., Schneider, F.B., 2008. Hyperproperties. In: Proceedings of the 21st IEEE Computer Security Foundations Symposium, CSF 2008, Pittsburgh, Pennsylvania, USA, 23-25 June 2008, IEEE Computer Society, pp. 51–65. https://doi.org/10.1109/CSF.2008.7.
Deb, S.B., Chetry, A., 2015. Usb device forensics: Insertion and removal timestamps of usb devices in windows 8. In: 2015 International Symposium on Advanced Computing and Communication (ISACC), pp. 364–371. https://doi.org/10.1109/ISACC.2015.7377371.
Debinski, M., Breitinger, F., Mohan, P., 2019. Timeline2gui: a log2timeline CSV parser and training scenarios. Digit. Invest. 28, 34–43. https://doi.org/10.1016/j.diin.2018.12.004. URL: https://www.sciencedirect.com/science/article/pii/S1742287618303232.
Du, X., Le, Q., Scanlon, M., 2020. Automated artefact relevancy determination from artefact metadata and associated timeline events. In: 2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), pp. 1–8. https://doi.org/10.1109/CyberSecurity49315.2020.9138874.
Dusane, P., Sujatha, G., 2022. Events of interest extraction from forensic timeline using natural language processing (nlp). In: Manogaran, G., Shanthini, A., Vadivu, G. (Eds.), Proceedings of International Conference on Deep Learning, Computing and Intelligence. Springer Nature Singapore, Singapore, pp. 83–94.
Farmer, D., Venema, W., 2004. Forensic Discovery. Addison-Wesley.
Gladyshev, P., Patel, A., 2005. Formalising event time bounding in digital investigations. Int. J. Digit. EVid. 4. URL: http://www.utica.edu/academic/institutes/ecii/publications/articles/B4A90270-B5A9-6380-68863F61C2F7603D.pdf.
Guðjónsson, K., 2010. Mastering the Super Timeline with Log2timeline. URL: https://www.sans.org/white-papers/33438/.
Groß, T., Dirauf, R., Freiling, F.C., 2020. Systematic analysis of browser history evidence. In: 13th International Conference on Systematic Approaches to Digital Forensic Engineering, SADFE 2020, New York, NY, USA, May 15, 2020, IEEE, pp. 1–12. https://doi.org/10.1109/SADFE51007.2020.00010, 10.1109/SADFE51007.2020.00010.
Hargreaves, C., Patterson, J., 2012. An automated timeline reconstruction approach for digital forensic investigations. Digit. Invest. 9, S69–S79. https://doi.org/10.1016/j.diin.2012.05.006. https://www.sciencedirect.com/science/article/pii/S174228761200031X. the Proceedings of the Twelfth Annual DFRWS Conference.
Harris, E.C., 1989. Principles of Archaeological Stratigraphy, second ed. Academic Press.
Henseler, H., Hyde, J., 2019. Technology assisted analysis of timeline and connections in digital forensic investigations. In: Proceedings of the First International Workshop on

AI and Intelligent Assistance for Legal Professionals in the Digital Workplace (LegalAIIA 2019), Hogeschool Leiden, Canada. URL: https://surfsharekit.nl/public/f9e4cd16-0577-4bb9-a41c-359ebecbf344.

James, J.I., Gladyshev, P., 2015. Automated inference of past action instances in digital investigations. In: Int. J. Inf. Secur. 14, 249–261. URL: 10.1007/s10207-014-0249-6 doi:10.1007/s10207-014-0249-6.

Kreibich, J.A., 2010. Using SQLite: small. Fast. Reliable. Choose Any Three. O'Reilly.

Lamport, L., 1978. Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21, 558–565. URL:10.1145/359545.359563 doi:10.1145/359545.359563.

Lee, R., 2012. Digital forensic sifting: colorized super timeline template for log2timeline output files. URL:https://www.sans.org/blog/digital-forensic-sifting-colorized-super-timeline-template-for-log2timeline-output-files/.

Lee, W.Y., Kim, K.H., Lee, H., 2019. Extraction of creation-time for recovered files on windows FAT32 file system 9, 5522. doi:10.3390/ap.9245522. URL: https://www.mdpi.com/2076-3417/9/24/5522.

Li, Q., Zhang, Q., Tan, Y.a., Li, Y., Zheng, J., 2016. Research on allocator strategy of FAT32 file system based on linux & windows. Atlantis Press, pp. 248–252. URL: https://www.atlantis-press.com/proceedings/icca-16/25847716. doi: 10.2991/icca-16.2016.58. ISSN: 2352-538X.

Levett, C.P., Jhumka, A., Anand, S.S.. Towards event ordering in digital forensics. URL: https://dl.acm.org/doi/10.1145/1854229.1854238.

Kaart, M., Laraghy, S., . Android forensics: Interpretation of timestamps 11, 234–248. URL: https://www.sciencedirect.com/science/article/pii/S1742287614000449, doi:10.1016/j.diin.2014.05.001..

Magnet Forensics, . How to use timeline in magnet axiom. https://www.magnetforensics.com/resources/how-to-use-timeline-in-magnet-axiom/. Accessed: 2024-01-24..

Mahalik, H., 2021. How to use the timeline features in cellebrite physical analyzer. https://cellebrite.com/en/how-to-use-the-timeline-features-in-cellebrite-physical-analyzer/, 2024-01-24.

Marrington, A., Baggili, I., Mohay, G., Clark, A., 2011. Cat detect (computer activity timeline detection): a tool for detecting inconsistency in computer activity timelines. Digit. Invest. 8, S52–S61. https://doi.org/10.1016/j.diin.2011.05.007. URL: https://www.sciencedirect.com/science/article/pii/S1742287611000314. the Proceedings of the Eleventh Annual DFRWS Conference.

Metz, J., 2018. Firefox cache file format. https://github.com/libyal/dtformats/blob/main/documentation/Firefox.

Metz, J., 2021. Pearls and pitfalls of timeline analysis. https://osdfir.blogspot.com/2021/10/pearls-and-pitfalls-of-timeline-analysis.html.

Minkowski, H., 1909. Raum und Zeit. Phys. Z. 10, 75–88.

Olsson, J., Boldt, M., 2009. Computer forensic timeline visualization tool. Digit. Invest. 6, S78–S87.

Patterson, J., Hargreaves, C., 2012. The Potential for Cross-Drive Analysis Using Automated Digital Forensic Timelines.

Plaso documentation, . Parsers and plugins. URL: https://plaso.readthedocs.io/en/latest/sources/user/Parsers-and-plugins.html.

Renfrew, C., Bahn, P.G., 1991. Archaeology: Theories, Methods and Practice. Thames and Hudson.

Schatz, B., Mohay, G., Clark, A., 2006. A correlation method for establishing provenance of timestamps in digital evidence. Digit. Invest. 3, 98–107. https://doi.org/10.1016/j.diin.2006.06.009.

Studiawan, H., Sohel, F., 2021. Anomaly detection in a forensic timeline with deep autoencoders. J. Inf. Secur. Appl. 63, 12. https://doi.org/10.1016/j.jisa.2021.103002. URL: https://www.sciencedirect.com/science/article/pii/S2214212621002076.

Studiawan, H., Sohel, F., Payne, C., 2020. Sentiment analysis in a forensic timeline with deep learning. IEEE Access 8, 60664–60675. https://doi.org/10.1109/ACCESS.2020.2983435.

Tse, K.W.h., 2011. Forensic analysis using FAT32 file cluster allocation patterns. pages 991032316259703414, b46605733. doi: 10.5353/th_b4660573. URL: https://hdl.handle.net/10722/143258.

Vaticle Ltd, 2024a. Meet typedb and typeql. https://typedb.com/features.

Vaticle Ltd, 2024b. Typedb: the polymorphic database powered by types. https://github.com/vaticle/typedb.

Weil, M.C., 2002. Dynamic time & date stamp analysis. International Journal of Digital. Evidence URL: https://www.semanticscholar.org/paper/Dynamic-Time.

Willassen, S.Y., 2008a. Finding evidence of antedating in digital investigations. In: 2008 Third International Conference on Availability, Reliability and Security, pp. 26–32. https://doi.org/10.1109/ARES.2008.149.

Willassen, S.Y., 2008b. Finding evidence of antedating in digital investigations. IEEE Computer Society. In: Proceedings of the the Third International Conference on Availability, Reliability and Security, ARES 2008, March 4-7, 2008. Technical University of Catalonia, Barcelona , Spain, pp. 26–32. https://doi.org/10.1109/ARES.2008.149, 10.1109/ARES.2008.149.

Willassen, S.Y., 2008c. Hypothesis-based investigation of digital timestamps. In: Ray, I., Shenoi, S. (Eds.), Advances in Digital Forensics IV, Fourth Annual IFIP WG 11.9 Conference on Digital Forensics, Kyoto University, Kyoto, Japan, January 28-30, 2008. Springer, pp. 75–86. https://doi.org/10.1007/978-0-387-84927-0_7, 10.1007/978-0-387-84927-0_7.

Willassen, S.Y., 2008d. Timestamp evidence correlation by model based clock hypothesis testing. In: Sorell, M., White, L. (Eds.), 1st International ICST Conference on Forensic Applications and Techniques in Telecommunications, Information and Multimedia, E-FORENSICS 2008, Adelaide, Australia, January 21-23, 2008, ICST/ACM, p. 15. https://doi.org/10.4108/e-forensics.2008.2637.

Willassen, S.Y., 2008e. Using simplified event calculus in digital investigation. In: Wainwright, R.L., Haddad, H. (Eds.), Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara, Brazil, March 16-20, 2008. ACM, pp. 1438–1442. https://doi.org/10.1145/1363686.1364020, 10.1145/1363686.1364020.

Willassen, S.Y., 2009. A model based approach to timestamp evidence interpretation. Int. J. Digital Crime Forensics (IJDCF) 1, 1–12. URL:. https://doi.org/10.4018/jdcf.2009040101, 10.4018/jdcf.2009040101.

Wilson, C., 2010. Manual identifiaction of suspect computer time zone. https://www.digital-detective.net/time-zone-identification/.