3-2019

Timeline2GUI: A Log2Timeline CSV Parser and Training Scenarios

Mark Debinski
University of New Haven

Parvathy Mohan
University of New Haven

Frank Breitinger
University of New Haven, fbreitinger@newhaven.edu

Publisher Citation

Debinski, Mark, Frank Breitinger, and Parvathy Mohan. "Timeline2GUI: A Log2Timeline CSV parser and training scenarios." Digital Investigation (2018). Volume 28, March 2019, Pages 34-43.

Comments

This is the authors' accepted version of the article published in *Digital Investigation.* The version of record can be found at http://dx.doi.org/10.1016/j.diin.2018.12.004

Timeline2GUI: A Log2Timeline CSV Parser and Training Scenarios

Mark Debinski, Frank Breitinger[*], Parvathy Mohan

Cyber Forensics Research and Education Group (UNHcFREG)
Tagliatela College of Engineering, ECECS
University of New Haven, 300 Boston Post Rd., West Haven CT, 06516

Abstract

Crimes involving digital evidence are getting more complex due to the increasing storage capacities and utilization of devices. Event reconstruction (i.e., understanding the timeline) is an essential step for investigators to understand a case where a prominent tool is Log2Timeline (a tool that creates super timelines which is a combination of several log files and events throughout a system). While these timelines provide great evidence and help to understand a case, they are complex and require tools as well as training scenarios. In this paper we present *Timeline2GUI* an easy-to-use python implementation to analyze CSV log files create by Log2Timeline. Additionally, we present three training scenarios – beginner, intermediate and advanced – to practice timeline analysis skills as well as familiarity with visualization tools. Lastly, we provide a comprehensive overview of tools.

Keywords: Log2Timeline, timeline, timestamps, parser, Timeline2GUI, training cases.

1. Introduction

Investigations become more complex due to amounts of data, connectivity, complexity of systems and encryption which causes backlogs in investigation bureaus (Quick & Choo, 2014). As a consequence, examiners use event reconstruction a.k.a. super timeline analysis – an approach that scans entire systems and combines all log file information into a single, comprehensive timeline. While these timelines are complex (may have millions of events), they are also a great resource (Chabot et al., 2014) and hard to manipulate, e.g., a single event like connecting a USB device usually cause multiple log entries in various files (Ieong, 2006).

One of the most prominent tools for super timeline creation is *Log2Timeline* which is an open source python implementation (details are described in Sec. 2.2). Nowadays it is also called *plaso* which is a rewrite of the predecessor (Gudjonsson, 2010b; Metz, 2017).

Problem Statement. While Log2Timeline is a powerful tool and can be helpful for investigations, there are currently major limitations:

1. There is no easy-to-use tool that beginners / investigators can use to analyze a generated timeline. According to our research, a common procedure is to convert the timeline (a.k.a. plaso storage file) into a CSV file using the psort.py commandline tool and then import the timeline into an Excel template although there are more powerful tools. Details about visualization tools are given in Sec. 2.4.
2. During our searches, we did not find free training material that allows practitioners to learn and improve their familiarity with Log2Timeline as well as visualization tools. To the best of our knowledge, the most utilized source is a Youtube video[1] posted by SANS titled SANS DFIR WebCast - Super Timeline Analysis with over 11,000 views. To counteract, it requires a set of well documented cases (log files + scenario descriptions) that everyone can use.

Contribution. We developed *Timeline2GUI* a standalone tool written in python (see Sec. 3) that supports the analysis of the CSV timeline (output from Log2Timeline). The tool is similar to the commonly utilized Excel sheet to allow an easy transition for practitioners. Additionally, three sample training cases (see Sec. 4) including solutions were created and are discussed in this article. Lastly, this article provides a thorough literature review of Digital Forensics Timeline Analysis including Visualization tools.

2. Related work

In the following subsections we first briefly talk about timeline analysis in general followed by a more detailed

[*] Corresponding author.
Email addresses: `MarkJDebinski@gmail.com` (Mark Debinski), `FBreitinger@newhaven.edu` (Frank Breitinger), `pmoha4@unh.newhaven.edu` (Parvathy Mohan)
URL: `http://www.unhcfreg.com/` (Mark Debinski), `http://www.FBreitinger.de` (Frank Breitinger)

[1] `https://www.youtube.com/watch?v=C4jNfXZ90fw` (last accessed 2018-12-05).

description of Log2Timeline. Next, we will highlight other timeline tools as well as visualization for timelines. The last subsection summarizes some challenges of timeline analysis.

2.1. Timeline analysis in general

According to Harrell (2011), "timeline analysis is great to determine when something has occurred at a certain time on a system." Thus, "creating a timeline of the various events that occurred during an incident is one of the key tasks performed by a digital forensic practitioner" (Esposito & Peterson, 2013). Chandrawanshi & Gupta (2013) even go a step further and say "logs offer an endless well of valuable information about systems, networks, and applications. Through logs, audit records, and alerts, information systems often give signs that something is broken (or 'broken into') or will be broken soon."

From a general perspective, timeline analysis consists of capturing system and network events and put them in order where events can be anything from file creation, file deletion, file modification, browser history, file transfers, account logins, and much more. Some literature differentiates between super, micro and nano / mini timeline analysis, depending on the amount of logs / events analyzed. For instance, "there are times when you don't want (or need) a super timeline, but instead just want to focus on one piece of available data, such as Event Log entries or Registry key LastWrite times" (Carvey, 2011). According to Carvey (2015), a micro timeline is created from either limited data sources and parsing a single log file can be considered as nano / mini timeline analysis. The benefits of small timelines are the runtime efficiency (they are fast to produce), less overwhelming and easier to analyze. On the other hand, a super timeline usually captures all system events[2]. A detailed discussion about super timeline analysis is provided by Esposito (2012) in the article 'Analysis of Forensic Super Timelines'.

2.2. Log2Timeline & Plaso

Log2Timeline was first introduced by Gudjonsson (2010b)[3] and is a super timeline analysis tool written in perl. In 2011, plaso was released as a rewrite of Log2Timeline; plaso being the name of the backend and Log2Timeline being the frontend. Wiedeman (2016) wrote "plaso has evolved from a single Perl tool, called Log2Timeline, to an 'engine' that amalgamates a number of useful forensic processes to produce super timelines." Metz (2017) describes it as "a framework that supports adding new parsers or parsing plug-ins, adding new analysis plug-ins, and writing one-off scripts to automate repetitive tasks in computer forensic analysis or equivalent." An overview of the parsers is provided in Appendix A. By the time writing this article, plaso / Log2Timeline is primarily maintained by Joachim Metz and Daniel White (see Github contribution history[4]) and can be used on MAC OS, Windows, and Linux.

When Log2Timeline is executed, it creates a plaso storage file (a container for the events / logs / timestamps) which then can be further processed using tools included into the plaso framework. The four major tools are:

Log2Timeline: As mentioned Log2Timeline is the frontend. Gudjonsson (2015a) describes it as "a command line tool to extract events from individual files, recursing a directory (e.g., mount point) or storage media image or device. Log2Timeline creates a plaso storage file which can be analyzed with the pinfo and psort tools. The plaso storage file contains the extracted events and various metadata about the collection process alongside information collected from the source data. It may also contain information about tags applied to events and reports from analysis plugins."

pinfo: This is a command line tool providing the user with information about what is stored in the plaso storage file. For instance, it provides information gathered during pre-processing, metadata on each storage container, the parsers used, the amount of extracted events or if there are tagged events. For more information see Gudjonsson (2015b).

psort: Filtering and sorting the plaso storage file, e.g., looking for events that happened during a particular timeframe, can be done using psort. Additionally, this allows to convert the plaso file into more common file formats such as CSV (Metz, 2015). A sample command to covert the plaso file into a CSV: `psort.py -o l2tcsv -w [CSV file] [Plaso file]` A full list of all output formats is provided in Appendix B.

- The *CSV timeline* stores 17 fixed fields as listed in Table 1 (SANS Institutes (2011) cheatsheet).

image_export: According to its Github page[5], "image_export is a command line tool to export file content from a storage media image or device based on various filter criteria, such as extension names, filter paths, file format signature identifiers, file creation date and time ranges, etc."

[2] Note, capturing all system events is impossible as it requires hundreds of thousands of different parsers, however, common super timeline tools include parsers for major events / logs such as browser history, file creation or system logs.

[3] Actually, the first release of Log2Timeline was in 2009 as indicated by the changelog: https://github.com/kiddinn/Log2Timeline/blob/master/CHANGELOG

[4] https://github.com/Log2Timeline/plaso (last accessed 2018-12-05).

[5] https://github.com/Log2Timeline/plaso/wiki/Using-image_export (last accessed 2018-12-05).

Table 1: Fields in the CSV file created by `psort.py`.

Field	Explanation
date	date of when the event occurred
time	time of when the even occurred
timezone	timezone that was used to call the tool with
MACB	Modification, Access, Creation, and Birth
source	source short name such as registry entries are REG
sourcetype	description of the source
type	timestamp type such as last accessed or last written
user	what user name is associated with event if any
host	what hostname is associated with entry is there is one
short	this contains a short description field where text is stored
desc	this is where majority of the information that is parsed is stored
version	gives the version number of the timestamp
inode	gives the inode number of the file being parsed
notes	additional storage location for information for some input modules
format	input module which was used to parse
extra	parsed information that is joined together and stored here. All these pieces of information make up the super timeline that Log2Timeline creates.

2.3. Timeline Creation and Analysis Tools

Besides Log2Timeline, there are other tools available for timeline analysis (free and commercial). Carbone & Bean (2011) "have compiled a representative list of digital forensic software tools and frameworks offering timeline generation capabilities in use today by forensic investigators" which include

- EnCase, FTK
- Log2Timeline
- The Sleuth Kit (ils, fls, mactime), PTK, Autopsy, Fiwalk
- Ex-Tip
- NTI FileList Pro
- Zeitline (Buchholz & Falk, 2005)
- AnalyzeMFT.py
- Mac-robber
- DFF (Digital Forensic Framework)
- The Coroner's Toolkit (grave-robber, mactime)
- NFILabs Aftertime
- SIMILE Timeplot

While we will not discuss these tools in detail, Carbone & Bean (2011) "provided a short timeline capability for each tool listed [...to] better comprehend their timeline generation capabilities." Some more information about timeline creation and analysis for different tools is also provided by Chapin (2013).

However, according to Eichelberger (2014), "each of these tools has the capability to generate a timeline based on the information collected from a system. From the above list, Log2Timeline provides the most diverse assortment of artifact information collection currently available using an automated straightforward implementation."

2.4. Visualization of timelines

Timelines, especially super timelines, have a large number of events (hundreds of thousands or even millions) which makes them "difficult to analyse and extremely problematic to visualise in a manner that is useful" (Hargreaves & Patterson, 2012).

The most native approaches to analyze the super timeline are commandline tools such as `mactime` from the Sleuthkit[6] or `grep`. While these tools are powerful, they are not user friendly and require proficiency. Similarly, examiners may use common text processing tools such as Excel, Apples Numbers, Wordpad or any other editor which may work for small timelines but is cumbersome for complex ones. Note, a common procedure to analyze Log2Timeline CSV files is a modified Excel template sheet[7] that highlights certain events (Weber, 2017).

Coming to general visual tools, one possibility is SIMILE Widgets Timeline[8] which creates a horizontal timeline that displays logs based on the time they have occurred. "While this seems like an ideal way to view Log2Timeline's output, there were many cautions against using this tool when there are more than 200 items as the SIMILE widget loads and operates very slowly"[9] (Esposito & Peterson, 2013). A similar tool, and thus also not favorable for large timelines, is BeeDocs (Gudjonsson, 2010a) (which is only available for Mac OS).

This last category are specialized tools; particularly made for forensic timeline analysis, log file analysis or specifically for Log2Timeine. The most well-known approach might be the ELK stack (Elasticsearch, Logstash

[6] `http://wiki.sleuthkit.org/index.php?title=Mactime` (last accessed 2018-12-05).

[7] `https://github.com/riodw/Log2Timeline-TIMELINE_COLOR_TEMPLATE` (last accessed 2018-12-05).

[8] `http://www.simile-widgets.org/timeline/` (last accessed 2018-12-05).

[9] While the original version around 2010 supported the SIMILE widget, the output format was removed from psort.py (Gudjonsson, 2010a).

& Kibana). While Log2Timeline supports exporting it to an Elasticsearch database and there are several tutorials online, we could only find a few usability reports. Walter (2016) mentions that he has "been pleasantly surprised at how easy it is to access timeline data, as well as perform searches using these tools." However, he misses "about the Excel process is the highlighting that visually called attention to items of interest." Furthermore, it also required some time to set up the complete system as well as needed some practice. The second tool is CyberForensics TimeLab (CFTL[10]) by Olsson & Boldt (2009) which shows a promising GUI however it does not seem to be maintained anymore - the last Github upload was over 6 years ago. On the other hand, it requires an XML file which is not supported by Log2Timeline anymore.

Timesketch is the third visualization possibility for Log2Timeline which is "an open source tool [... that was] designed to make collaboration, sharing and search easy as well as quickly correlate disparate events" and is mainly developed by Berggren et al. (2018) (most Github contributions).

Given that there is only limited information about the tool, the following assessment is based on our experiences using it. According to Timesketch's Github page, it offers three different types of installation: Install Timesketch manually, using docker or using Vagrant. The first attempt to use Timesketch on Windows 10 was troublesome as there are only instructions for Linux. The next option was to use Vagrant where we identified some complications when being used on Windows. The last option did not work as well since Docker only works on Windows 10 Pro Operating Systems. After switching to Mac OS, the Vagrant installation method worked smoothly. The installation process was simple as instructions were provided but it was time consuming.

Once installed, Timesketch has various features such as filters (e.g., set any time range they want to view) or charts (e.g., user has a option to see a heatmap or a histogram). Both options allow the user to see the timeline from different angels. In addition, Timesketch allows users to 'star' (similar to a e-mail flag) any logs which indicates a higher importance. Logs in Timesketch are also similarly highlighted like those in Timeline2GUI. While using TimeSketch, it was noticed that loading large CSV files took a long time for the program to process.

To sum it up: while the GUI is very similar to Timeline2GUI, Timesketch has more functionality but the installation is more complex and performance seems slower for large timelines.

Lastly, there is Evidence Fetcher (efetch) a web-based file explorer, viewer, and analyzer for timelines created by Maurer (2016). Similar to CFTL, it looks like a promising environment but seems not maintained as there was only few commits since October 2016.

To summarize: there are several tools for timeline analysis however several are outdated / not maintained anymore. The two most promising approaches are Timesketch and ELK Stack which come with a lot of functionality but also require configuration; they are not as straightforward to use as Timeline2GUI.

2.5. Challenges of Timeline analysis

There are several challenges of timeline analysis mainly discussed by Hargreaves & Patterson (2012) and Chabot et al. (2015). Both articles stress that it is difficult to analyze the large number of events (e.g., can be several million 'low-level' events) and problematic / important to visualize the information.

To support investigators, Hargreaves & Patterson (2012) "propose a technique that can automatically reconstruct high-level events (e.g. connection of a USB stick) from this set of low-level events." While their evaluation shows promising results and reduces the complexity / information overload of super timelines, the Python Digital Forensic Timeline (PyDFT) utilizes its own parser framework to extract events which replicates much of Log2Timeline. Thus, it is irrelevant for individuals using Log2Timeline.

Chabot et al. (2015) identified three major problems (volume, heterogeneity, and legal requirements) with event reconstruction and present seven necessary criteria (addressing the problems) that an efficient reconstruction tool must meet. With respect to volume, the authors argue that "many tools do not offer an intuitive interface but only a query tool that appears to be a powerful but complex and tedious way to access the information. This is the case of approaches using databases and providing a SQL query interface which is efficient but not intuitive for an untrained user." In detail, they propose the following three requirements:

1. Automation to reconstruct and analyze timelines and certain tasks that become too complex to be carried manually.
2. Visualization tools that highlight relevant information; guide investigators to interpret, analyze and draw conclusions.
3. Efficient browsing of the data in a clear and intuitive way.

With respect to heterogeneity, the authors stress that information is spread throughout systems and it is important to consider a broad variety of different sources. For legal requirements, "the challenge is to ensure that the results are admissible in a court of law." Note, given that Timeline2GUI is only a visual frontend, the requirements proposed for heterogeneity and legal requirements are less relevant.

Based on their identified criteria, they present a three-layered ontology, called ORD2I including several scripts to support investigators. However, their implementations suffer performance issues: it takes about 3h to process 20,000 entries (which is a very small timeline).

[10] https://github.com/jensolsson/CFTL (last accessed 2018-12-05).

3. Timeline2GUI Tool

Timeline2GUI is a graphical frontend that can read CSV files generated by Log2Timeline and supports their analysis. The goal was to make the parsing (reading) of the log files straightforward for the end user. The GUI is kept simple and is based on the Excel sheet which is widely used[11]. A screenshot can be found in Figure 1. Timeline2GUI can be download from Github: https://github.com/parvathycec/Timeline2GUI.

With respect to the previously mentioned requirements, Timeline2GUI highlights relevant information which can also be easily changed by a user (details see Sec. 3.1). For efficient browsing, we implemented two views (see Fig. 1) . *Reduced view* is a summary of major events, i.e., it shows all highlighted events; *detailed view* shows the full timeline. To switch from reduced to detailed view, one can use the index column. Our tool currently does not have any automation to keep it simple but more importantly performant.

3.1. Functionality

Timeline2GUI Version 1.0 allows loading a CSV file by clicking the 'Select CSV File' button. Next, the applications allows the user to filter the logs before actually loading them where filtering requires a date range (in the format YYYY-MM-DD HH:MM:SS), e.g., `date > '2017-03-16' and date < '2017-03-17'`. Optionally, one can add a time to make it even more precise, e.g., `date > '2017-03-16 12:34:56' and date < '2017-03-17 10:10:10'`. Once loaded, Timeline2GUI has an easy-to-use search functionality to look for a specific keywords which can be accessed by clicking the search button. Filtering or searching will only change the view but not delete / modify the loaded CSV file. Thus, a user can always revert to the complete timeline by pressing the load data button.

To support investigations, Timeline2GUI highlights log entries which could be relevant and thus makes skimming the timeline easier. Specifically the following highlighting is done where the logic is copied from the Excel template:

Green indicates that a file may have been opened or created.

Yellow shows that the event is related to web activity.

Blue indicates that an external device (e.g., USB stick) has been mounted / interacted with the system.

Red means some sort of execution was done on the system like a application was started.

Magenta highlighting indicates files that have been deleted.

Grey highlights indicate logs like for example firewall logs.

Note, color settings are stored in a configuration file and can be changed (details are in the next section). After the desired results are found, our tool can store the filtered logs into a new CSV file.

3.2. Implementation

Timeline2GUI is written in Python 3.6 and utilizes 3 major libraries: (1) Tkinter, (2) Pandas, and (3) Pandastable (developer version[12]). The data from the CSV file is imported to a Pandas DataFrame. Pandas DataFrame is an efficient data structure to store data from a spreadsheet and provides powerful data operations like filtering or searching. The pandastable library provides a table widget for Tkinter with plotting and data manipulation functionality. It uses pandas' DataFrame class to store table data. Hence, we have used this combination of libraries to display and manipulate data from CSV files. The operations performed by the tool are as follows:

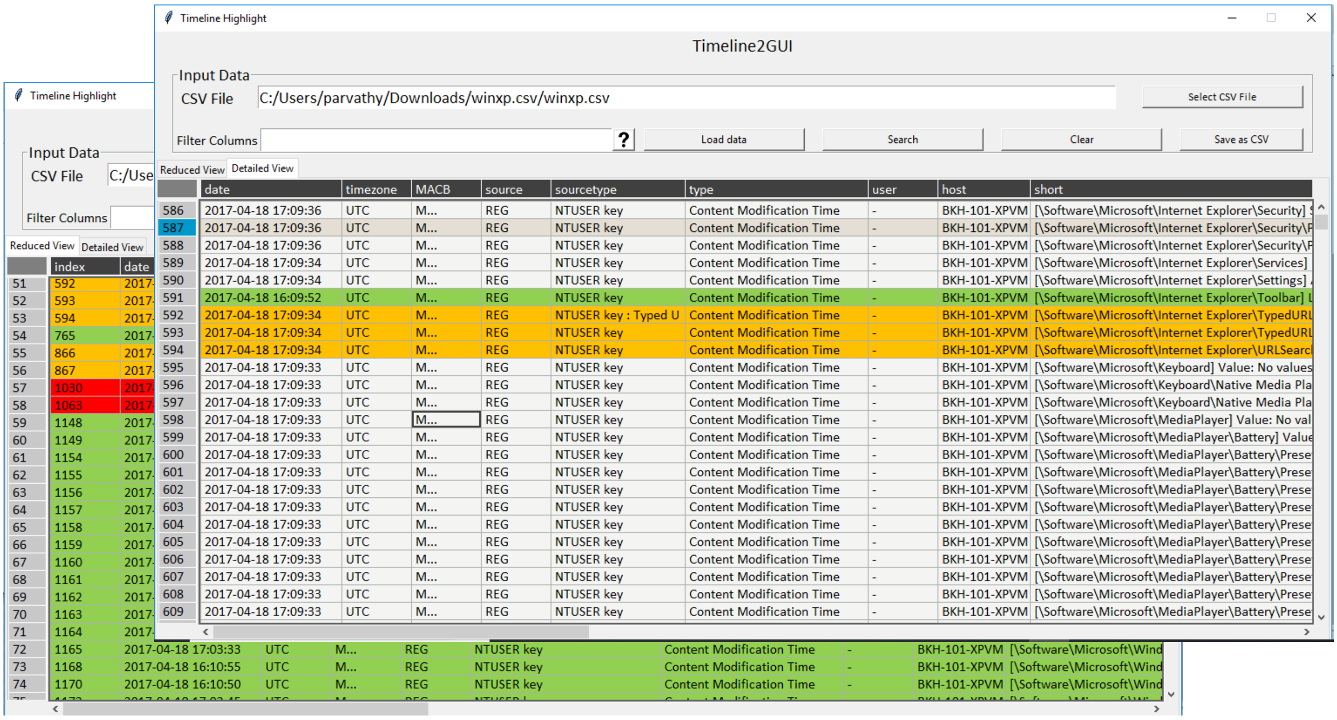
1. *Filter the data based on column values:* If a query is given in the filter value text field, only the rows of the CSV file which satisfies the query will be displayed. The DataFrame.query(query) function is used to filter the data. Query supports operators like >, <, == etc.
2. *Sort column values:* This feature allows the user to double click on the column header to sort the column values in ascending/descending. This feature was integrated with pandastable table widget.
3. *Free text search:* This feature allows a user to search for a string anywhere in the CSV data. Pandas DataFrame provides a 'contains' function to check if the given string is in any of the column / row.
4. *Highlight rows in the table widget:* The user can configure automatic highlighting in `highlights.txt` which helps finding certain events faster. The file accepts one setting / entry per line, e.g.,

   ```
   *=CONTAINS=USB=#1E00FF
   short=ENDS=.lnk=#92D051
   ```

   where each entry consists of four parts separated by the equal sign (=). The first part is the column name which can be one particular column (e.g., short) or * which indicates any column. The second parameter is the comparison operator which can be STARTS, ENDS, EQUALS or CONTAINS (indicating how the string will be searched). Third is the keyword we are looking for which can contain spaces or other special characters except the equal sign. The last option is the color code. Consequently, the two lines above mean:

[11] https://github.com/riodw/Log2Timeline-TIMELINE_COLOR_TEMPLATE (last accessed 2018-12-05).

[12] http://pandastable.readthedocs.io/en/latest/ (last accessed 2018-12-05).



C:/Users/parvathy/Downloads/winxp.csv/winxp.csv

	date	timezone	MACB	source	sourcetype	type	user	host	short
586	2017-04-18 17:09:36	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\Internet Explorer\Security]
587	2017-04-18 17:09:36	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\Internet Explorer\Security\F
588	2017-04-18 17:09:36	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\Internet Explorer\Security\F
589	2017-04-18 17:09:34	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\Internet Explorer\Services]
590	2017-04-18 17:09:34	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\Internet Explorer\Settings]
591	2017-04-18 16:09:52	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\Internet Explorer\Toolbar] L
592	2017-04-18 17:09:34	UTC	M...	REG	NTUSER key : Typed U	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\Internet Explorer\TypedURL
593	2017-04-18 17:09:34	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\Internet Explorer\TypedURL
594	2017-04-18 17:09:34	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\Internet Explorer\URLSearcl
595	2017-04-18 17:09:33	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\Keyboard] Value: No values
596	2017-04-18 17:09:33	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\Keyboard\Native Media Pla
597	2017-04-18 17:09:33	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\Keyboard\Native Media Pla
598	2017-04-18 17:09:33	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\MediaPlayer] Value: No val
599	2017-04-18 17:09:33	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\MediaPlayer\Battery] Value
600	2017-04-18 17:09:33	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\MediaPlayer\Battery\Prese
601	2017-04-18 17:09:33	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\MediaPlayer\Battery\Prese
602	2017-04-18 17:09:33	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\MediaPlayer\Battery\Prese
603	2017-04-18 17:09:33	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\MediaPlayer\Battery\Prese
604	2017-04-18 17:09:33	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\MediaPlayer\Battery\Prese
605	2017-04-18 17:09:33	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\MediaPlayer\Battery\Prese
606	2017-04-18 17:09:33	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\MediaPlayer\Battery\Prese
607	2017-04-18 17:09:33	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\MediaPlayer\Battery\Prese
608	2017-04-18 17:09:33	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\MediaPlayer\Battery\Prese
609	2017-04-18 17:09:33	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\MediaPlayer\Battery\Prese

	index	date								
51	592	2017								
52	593	2017								
53	594	2017								
54	765	2017								
55	866	2017								
56	867	2017								
57	1030	2017								
58	1063	2017								
59	1148	2017								
60	1149	2017								
61	1154	2017								
62	1155	2017								
63	1156	2017								
64	1157	2017								
65	1158	2017								
66	1159	2017								
67	1160	2017								
68	1161	2017								
69	1162	2017								
70	1163	2017								
71	1164	2017								
72	1165	2017-04-18 17:03:33	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\Wind
73	1168	2017-04-18 16:10:55	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\Wind
74	1170	2017-04-18 16:10:50	UTC	M...	REG	NTUSER key	Content Modification Time	-	BKH-101-XPVM	[\Software\Microsoft\Wind

Figure 1: Screenshot of our Timeline2GUI application with a CSV loaded. The image in the background shows the *reduced view* and the image in the foreground shows the *detailed view*. The index column in the reduced view and be used to find the corresponding section in the detailed view.

- Highlight any row in blue (hex code #1E00FF) that contains the string 'USB'.
- Highlight any row in green (hex code #92D051) where the string in the short-column ends with '.lnk'.

Note, keywords are case *insensitive*. To highlight multiple keywords with the same color, one may add another entry with the same color code, e.g., `*=CONTAINS=SanDisk=#1E00FF`.

5. *Save the displayed data to a CSV file*: Pandas DataFrame contains a method to_csv(filename, index, encoding) to save the current displayed information to a CSV file.

```
self.current_data.to_csv(filename,
          index=False, encoding='utf-8');
```

3.3. Testing

Given that Timeline2GUI only changes the representation of the timeline (from CSV to Python/Pandas), testing was primarily accomplished by using the tool for solving the training cases. Specifically, we focused on making sure that query filtering was working properly and that the displayed information was correct.

4. Sample Training Cases

In total, three cases were developed which rank from beginner over intermediate to advanced. The timelines in CSV format can be downloaded from `https://www.fbreitinger.de/wp-content/uploads/2018/08/Timeline2GUI_training_cases.zip`.

For each scenario, we have brief *background stories* which describe the scenario and can be passed to a user / student / practitioner; these will set the expectations / baseline. Next, we describe the setup, i.e., some more technical details on how the scenario was created. Lastly, we present the solution for each case using Timeline2GUI (detailed view). Before presenting the details about the training cases, we summarize our overall apparatus.

Note, the CSV timelines are independent from Timeline2GUI and can be used with any other tool capable of parsing Log2Timeline CSV files.

4.1. Training cases setup

To create super timelines for the training cases, Log2Timeline was installed on MAC OSX and multiple virtual machines (VirtualBox) running Microsoft Windows 7 were created. Each VM was then prepared as discussed in the corresponding subsections (see 4.2.2, 4.3.2 and 4.4.2). After creating the cases, we extracted the timelines running the following command: `Log2Timeline.py scenario1.plaso scenario.vhd` which creates a plaso file named 'scenario1.plaso' from the virtual hard disk (vhd). In a final step, we executed `psort` to convert the plaso file into the CSV format. The command used was `psort.py -o l2tcsv -w scenario1.csv` `scenario.plaso`. The procedure was repeated for all three cases.

For the second scenario, we additionally created logs for a USB device which is part of the case. Therefore, we first mirrored the USB device using `sudo dd if=/dev/disk2 of=/Users/Desktop/USBDrive.dd` and then proceeded as before running (`Log2Timeline.py USBDrive.plaso USBDrive.dd`) followed by `psort.py -o l2tcsv -w usb2sen.csv USBDrive.plaso`.

4.2. Case I - Beginner

The beginner case simulates a computer crime that occurred to an office employee. The following three subsections include the case description, the setup and lastly the solution, respectively.

4.2.1. Case description

Hannah, an employee for a midsize company, received an email from the IT-Administrator containing a password in an attached word file. Hannah downloaded the file called 'password.rtf' from her Gmail account and stored it into a folder called 'secret' on the desktop (directory). After downloading and storing the file, Hannah continued browsing on the Internet before she walked away from her workstation to answer a phone call on her cellphone.

A few days later Hannah believes someone stole the password that was given to her in the email because she received a notification of a login attempt. Hannah told the IT department that she believes the file was stolen from her computer while she was on the phone. The phone logs indicate that the call was on 2018-02-29 at 21:10:00 and only lasted for 5 minutes. Hannah is unsure if she locked the computer when she left it. Can you help to figure out what happened?

4.2.2. Setup

The scenario was created within a 1 - 2 hour span on the virtual machine. Google Chrome was downloaded and set up so Gmail can be accessed in order to download the password file. Specific directories were set up throughout the virtual machine that were used to store files. A USB drive was utilized to copy the secret folder from the computer (the USB device / timeline is not available).

4.2.3. Solution

To solve the scenario, Timeline2GUI was used to parse and search the log files. The key elements from the description are: date and time, password.rtf and secret folder.

Note: we provide the line numbers where a particular event was found. Although they should be identical if you follow our solution, there might be minor changes.

1. In the filter columns bar the query (date > '2018-03-29 21:00:00' and date < '2018-03-29 21:15:00') was entered to reduce the timeline to the relevant time frame. We used a longer time frame (15 min) in case the device time was not synchronized.
2. Next, we utilized the search bar text field to look for 'secret' (this allows us to get closer to the actual event).
3. Under the refreshed results log number 0 shows that the 'secret' folder was created at 21:02:12.
4. Log number 13 shows content modification because the file 'password.rtf' was created in the 'secret' folder at 21:03:46.
5. Log number 31 shows that the 'secret' folder was accessed at 21:05:04 which is highlighted in green (this flow coincides with Hannah's description).
6. Due to the 'secret' folder being last accessed at 21:05:04, changing the query (filter column) to (date > '2018-03-29 21:05:00' and date < '2018-03-29 21:15:00') will eliminate older logs.
7. Next, we continued scrolling until we found lines highlighted in yellow indicating browser / Internet activity (around line 2575 at 21:08:55). This is when Hannah walked away from the computer to answer the call (update filter to date > '2018-03-29 21:08:55' and date < '2018-03-29 21:15:00').

We now know the exact time and can focus on what happened which is the more challenging part. Timeline2GUI is set up to color events automatically, e.g., web activity, prefetch files, etc. While our solutions are based on the *detailed view*, we advice to start with the reduced view and focus on those events.

8. Scrolling down further reveals a line marked blue (line 255) which means USB activity. Specifically, a SanDisk USB drive was plugged in at 21:10:14.
9. Line 397 is marked red which shows a WinPrefetch[13] log at 21:10:31. This indicates that WUDFHOST.exe was executed on the system. WUDFHOST.exe is a process used by windows to load drivers and to communicate with hardware devices[14].
10. Line 447 indicates another WinPrefetch log that occurs at 21:10:41 (marked red) which points to EXPLORER.EXE. Usually EXPLORER.EXE is executed from the GUI when a user opens or writes to drives and folders[15].
11. At 21:14:31 there is the next Google Chrome activity which is an indicator that Hannah returned to her device.

To summarize the results: it looks like Hannah did not lock her device after she left her PC and someone plugged in a USB dongle (SanDisk) and performed some actions. Although there is no clear indication in the log files that the secret folder / file was copied onto the USB (this would

[13] According to McQuaid (2014), Windows operating systems create prefetch files when applications are executed on the system.

[14] `http://www.processlibrary.com/en/directory/files/wudfhost/78364/` (last accessed 2018-12-05).

[15] `http://www.processlibrary.com/en/directory/files/explorer/24746/` (last accessed 2018-12-05).

be found only on the USB device), someone clearly accessed the device while she was on the phone. Thus, making the assumption that the folder / file was stolen is compelling.

4.3. Case II - Intermediate

The intermediate scenario is a case where illegal files have been downloaded.

4.3.1. Case description

Bob is a suspect to police for downloading illegal files from 'DropBox'. Police apprehended a warrant and executed it on 2018/04/06 to seize any electronic devices in his house. Authorities collected multiple devices that belonged to Bob to find evidence that was downloaded. Investigators examined all the devices collected and narrowed it down to a computer and a USB drive. The computer did not contain any downloaded files; the USB drive has a file named secrets_file which cannot be accessed / opened. Investigators believe that it is encrypted because VeraCrypt was found on the device. Note: Police records indicate that the files were downloaded on 2018/04/05.

4.3.2. Setup

The scenario was created within a 2 day span on the virtual machine. We accessed the DropBox website and downloaded content using direct links. Additionally, we installed VeraCrypt on the system and created an encrypted container which contains the downloaded files. A USB drive was utilized to store the encrypted container (moved from the PC to the USB device).

4.3.3. Solution

The information used to solve this case was from the police reported which provides details on what to analyze (e.g., the existence of the secret_files).

1. Given the information from the case description, we used the following filter (date > '2018-04-05 00:00:00' and date < '2018-04-06 23:59:59') to narrow down results. Subsequently, we searched for 'dropbox'.
2. The refreshed logs reveal 'Classified.rtf', 'TopSecret.rtf', and 'EvidenceTopSecret.png' in lines 232, 240, 260, respectively, which were downloaded between 23:04:28 and 23:07:04 on 2018-04-05. The logs are highlighted in yellow (webhist) which was how we identified them.
3. Knowing the time range and file names, we removed the 'dropbox' filter (which brings us back to (date > '2018-04-05 00:00:00' and date < '2018-04-06 23:59:59')) and then searched for 'download'.
4. Lines 147 to 154 (yellow) show that VeraCrypt was downloaded at 14:18:48 on 2018-04-05.
5. In a next step we narrowed down the time range to (date > '2018-04-05 23:07:05' and date < '2018-04-06 23:59:59') which hides all activities that occurred prior to downloading 'EvidenceTopSecret.png' (compare item 2). The yellow line 3053 at 22:18:35 then revealed that Bob had visited a website that gives the user a beginners tutorial on how to use VeraCrypt. This coincides with the investigator's assumption that VeraCrypt may have been used.
6. Next, we set (date > '2018-04-06 22:18:36' and date < '2018-04-06 23:59:59') to focus on the time after the VeraCrypt tutorial was searched. The red line 1075 shows that VERACRYPT FORMAT.EXE was executed.
7. Going down a bit further to line 1161 shows that a secret_files was created (note the log entry can be both file or folder creation). Explanation: The 'short' column of the log states USB_REASON_FILE_CREATE which means that a file or directory was created for the first time[16].
8. About 200 entries later, line 1361 indicates that secret_files has been overwritten and the file's basic information was changed. According to Microsoft[16], USN_REASON_DATA_OVERWRITE means that data has been changed in the file / directory and USN_REASON_BASIC_INFO_CHANGE means that a user may have changed the file / directory attributes or timestamps.
9. Subsequently, logs 1372 to 1380 show that the downloaded files TopSecret.rtf, Classified.rtf, and EvidenceTopSecret.png had security changes. Explanation: The logs showed USN_REASON_SECURITY_CHANGE which means that access permissions have been changed to the file or directory[16].
10. Logs 1383, 1385, and 1386 which occur at 23:20:51 on 2018-04-06 show that the three files were deleted from the system (logs show USN_REASON_FILE_DELETE[16]).

Summarizing all the logs / events, one may assume that the suspect created a container using VeraCrypt named secret_files and copied the files into it. Afterwards, the suspect deleted the files off the device.

11. The blue log 1430 (22:36:09 on 2018-04-06) shows the first indication that a USB Drive was plugged in the computer followed by some WinPrefetch activity in log 1448 at 22:36:25 (red) which shows that EXPLORER.EXE was executed (e.g, to allow Bob to access the files on the USB device). Next, log 1455 shows another WinPrefetch log loading WUDFHOST.exe (recall: this is the process to load drivers and communicate with hardware devices).
12. Continuing searching for evidence, we can see the red log 1531 which states that VeraCrypt was executed again on the system at 22:37:14.

[16] https://docs.microsoft.com/de-de/windows/desktop/api/winioctl/ns-winioctl-read_usn_journal_data_v0 (last accessed 2018-12-05).

13. The USN_REASON_BASIC_INFO_CHANGE value is shown for the secret_files container in log 1549 (22:37:19). This means that there is some type of activity with the file shortly after VeraCrypt was executed. Logs 1665 and 1666 indicate that something was deleted shortly after the secret_files folder was being modified (one can only see that something was deleted but there is no proof that it was the files downloaded).

The final step is to load the usb2sen.csv into Timeline2GUI which is the log file for the USB Drive. Due to small timeline, no filter is required.

14. Log 40 at 22:37:33 shows that the secret_files was created on the USB.

In summary, the logs support the assumption that VeraCrypt was executed and 'secret_files' was created. Furthermore, we saw that security changes were made to all downloaded files before they were deleted. Next, a USB Drive was connected where the USB logs showed that secrets_files was created around the same time. Once the file was moved to the USB, the secrets_file was deleted from the system. To conclude, there is a high chance that the files are located in the secrets_files container on the USB dongle. However, in order to be 100% certain, one would have to open the encrypted container or see if the files can be recovered (e.g., file carving).

4.4. Case III - Advanced

The advanced scenario is a case where a suspect's computer is seized, and then evidence is found that the suspect was stealing credit card information but the suspect claims s/he innocent.

4.4.1. Case description

Alice received a phone call from a person trying to steal her credit card information. Fortunately she was prepared and did not fall for the scam but contacted the police and provided the phone number that called her. The police tracked down the cellphone owner to be a person by the name Fred who was arrested on April 19th. Fred claims that he did not steal credit card information nor wanted he to sell them on the dark web. To the contrary, he claims that it was the previous owner from whom he purchased the computer and phone on April 4th. While seizing the computer, the machine was still running and the officers noticed that a TOR browser was open and that 'setMACE' was downloaded successfully.

4.4.2. Setup

The scenario was created over a 4 day span and is 34 GB in total (compared to the previous 8 GB images). Our intention was to create a more realistic and comprehensive case. Avast Internet Security was installed and security scans were executed periodically once a day (increased number of log entries). Additionally, TOR was installed for accessing the dark web as well as Google Chrome for regular browsing. Lastly, setMACE, which is an anti-forensic tool that allows to tamper with timestamps, was installed but never executed[17]. Several files and directories were set up throughout the virtual machine.

4.4.3. Solution

Based on the case description, we know that the Fred is accused of credit card fraud / scam and that the police arrested the suspect on April 19, 2018. Given that our initial filter will look at a wide range of events: (date > '2018-01-01 00:00:00' and date < '2018-04-20 23:59:59').

1. We started by searching terms like scam, credit and fraud which revealed several entries. Logs 0 to 2 indicate that a 'credit card info' file was created on the system at 14:31:00 on 2018-04-05 (lines are highlighted).
2. Having a better idea about the timeframe, we adjusted the filter to (date > '2018-04-05 14:20:00' and date < '2018-04-20 23:59:59'). The aforementioned creation of the file can now be found in logs 2389 and 2390.
3. While scrolling, two more green lines caught our attention: 'phone numbers.rtf' in line 2405 at 14:33:16 on 2018-04-05 followed by 'Call Instructions' (line 2671 at 14:36:16). Note, the files are included and can be analyzed as well.
4. Next, we looked for modifications of the files. Therefore, re-searching 'credit' revealed several logs ranging from 2018-04-06 to 2018-04-19 which indicate that the credit card info file was modified (e.g., maybe cause of adding new credit card details). The logs are highlighted in yellow and green.

Having a look at these 3 files (provided in the scenario.zip file) reveal that 'phone numbers.rtf' contains a list of phone numbers which was utilized by the 'Call Instructions' pseudo script to call the individuals. If successful, the data was stored in 'credit card info'.

Given that the logs prove that the files had been created after April 4th (the date the machine was purchased), it is unlikely that the previous owner was involved. Moving forward, the investigator can focus on the question if the suspect attempted to sell the information (charge him with attempting to sell stolen property).

6. To analyze web activity, we filtered for (source == 'WEBHIST') which returns what the user had searched / visited on the Internet.
7. One will find setMACE in log 4832 (on 2018-04-12 at 23:00:23; highlighted yellow) and a hint for TOR

[17] While there were several attempts to execute it, it never actually run.

browser in log 5013 (on 2018-04-19 at 23:04:10; highlighted yellow)[18].

8. We then went back to the complete range (date > '2018-04-05 14:20:00' and date < '2018-04-20 23:59:59') and searched for 'setMACE'.
9. Log 36 to 54 (on 2018-04-10 at approximately 14:34) show that setMACE was downloaded onto the computer.
10. Continuing scrolling, logs 300 and 338 allows to conclude that setMACE was executed on the system. Both of execution happened on 2018-04-19 (highlighted in red).
11. As a next step we focused on TOR since the browser was active when seizing the machine. Therefore, we searched for TOR and TorProject which led us to 2018-04-19 at 23:04:32 – TOR was downloaded.
12. Focusing on the final 2 days before the device was seized, the results revealed that Fred searched for how to access the dark web on 2018-04-19 at 23:03:24. A few seconds later (23:03:55) we can see that he searched for how to sell information online.
13. There were no identifiers / logs that the user actually accessed the dark web or sold information but he certainly informed himself.
14. Lastly, we had a second look at setMACE again where we focused on the last day. Under the refined results, logs 17, 23, 37, 42, 50, 55, 69, 82, 87, 98, 106, 114, and multiple more logs indicated that setMACE failed to run on the system. This is an indicator that setMACE was not executed correctly on the system by the user and it is likely that nothing was manipulated.
15. The save CSV button was pressed and all the logs for 2018-04-19 were saved because it was within that time frame where the majority of the actions happen.

To conclude this scenario: Some evidence was found on the device (e.g, telephone numbers, credit card numbers as well as a script for calling individuals) indicating that the machine was used for illegal activities. All logs imply that the activities happened after April 4th (the date the machine was purchased). Even though a timestamp manipulation tool (setMACE) was found, the timeline seems unaltered and does not contain any outliers. The existence of TOR and the search history show an intent to sell information but there is no evidence that it was sold or for accessing the dark web.

5. Conclusion and future work

Reading / parsing CSV files is straightforward and several tools exist (e.g,. Excel, Apples Numbers). However, research stresses the importance of dedicated tools that support investigators to do fast filtering and allow efficient browsing (e.g., highlighting improves the user experience and eventually accelerate the process).

This worked presented an easy-to-use parser for CSV timelines (created with Log2Timeline) named *Timeline2GUI* as well as three training cases – beginner, intermediate and advanced – including detailed descriptions of the setup and solutions. The layout and highlighting capabilities of Timeline2GUI are based on the well-known Excel sheet. However, our tool comes with two views (reduced view and detailed view), an enhanced filtering mechanism and is more straightforward to use. Additionally, our python tool is open-source and thus can be extended.

We developed three training cases that are freely available, independent from our implementation and can be used to improve investigator's timeline analysis skills by either using Timeline2GUI, the Excel sheet or any other tool. Our scenarios range from from 8 GB to 34 GB which can be seen rather small compared to the real world scenarios. The timelines (logs) are available for download (see Sec. 4); one possible solution path per case was provided in this article.

As indicated by our solutions, highlighting certain events (e.g., file creating, web history or prefetch files) certainly helps to find interesting / relevant events. However, in order to understand the complete case, an investigator also has to be familiar with computer / operating system events and may still have to look at x-thousand logs to find a clue. For instance, it is important to be familiar with system events, e.g., what are the implications if a prefetch log event occurs or if WUDFHOST.exe is run. In case of beginners, it may be helpful to create a cheat sheet that contains the log event and a brief explanation on its implication. Overall, reviewing the 3rd scenario took about 4-5 hours using Timeline2GUI (although we created the case). It would have taken longer when looking at the plain text using a regular text editor.

Timeline analysis is a powerful technique to understand what happened to a system or device. However, it is not trivial and can be very time consuming. In the future, we want to further improve Timeline2GUI to hopefully speed up the process, e.g., by removing irrelevant events, or combining events. In a first step, we will explore if we can use techniques from approximate matching combine similar events (Breitinger et al., 2014).

References

Berggren et al. (2018). Timesketch. https://github.com/google/timesketch.

Breitinger, F., Guttman, B., McCarrin, M., Roussev, V., & White, D. (2014). Approximate matching: definition and terminology. *NIST Special Publication*, *800*, 168.

Buchholz, F. P., & Falk, C. (2005). Design and implementation of zeitline: a forensic timeline editor. In *DFRWS*.

Carbone, R., & Bean, C. (2011). *Generating Computer Forensic Super Timelines under Linux: A Comprehensive Guide for Windows-based Disk Images*. Technical Report Defence Research and Defence Canada-Valcartier Quebec, Quebec.

[18] Note, we scrolled through and looked at it line by line but especially for highlighted lines.

Carvey, H. (2011). Howto: Creating mini-timelines. https://windowsir.blogspot.com/2011/09/creating-mini-timelines.html.

Carvey, H. (2015). Micro & mini-timelines. https://windowsir.blogspot.com/2015/04/micro-mini-timelines.html.

Chabot, Y., Bertaux, A., Nicolle, C., & Kechadi, M.-T. (2014). A complete formalized knowledge representation model for advanced digital forensics timeline analysis. *Digital Investigation*, *11*, S95–S105.

Chabot, Y., Bertaux, A., Nicolle, C., & Kechadi, T. (2015). An ontology-based approach for the reconstruction and analysis of digital incidents timelines. *Digital Investigation*, *15*, 83–100.

Chandrawanshi, R., & Gupta, H. (2013). A survey: Server timeline analysis for web forensics. *International Journal of Scientific Research Engineering and Technology (IJSRET)*, *1*, 017–021.

Chapin, B. (2013). *Timeline Creation and Analysis Guides*. Technical Report Senator Patrick Leahy Center for Digital Investigation (LCDI). https://www.champlain.edu/Documents/LCDI/Timeline_Creation_and_Analysis_Guides.pdf.

Eichelberger, F. (2014). Automation of report and timeline-file based file and url analysis. *Interested in learning more about cyber security training? SANS Institute InfoSec Reading Room*, .

Esposito, S., & Peterson, G. (2013). Creating super timelines in windows investigations. In *IFIP International Conference on Digital Forensics* (pp. 135–144). Springer.

Esposito, S. J. (2012). *Analysis of forensic super timelines*. Technical Report Air Force Inst Of Tech Wright-Patterson AFB OH School Of Engineering And Management.

Gudjonsson, K. (2010a). Mastering the super timeline - log2timline style. https://digital-forensics.sans.org/summit-archives/2010/eu-digital-forensics-incident-response-summit-kristinn-gudjonsson-mastering-the-super-timeline.pdf.

Gudjonsson, K. (2010b). *Mastering the super timeline with log2timeline*. Technical Report. https://www.sans.org/summit-archives/file/summit-archive-1493923574.pdf.

Gudjonsson, K. (2015a). Using log2timeline. https://github.com/log2timeline/plaso/wiki/Using-log2timeline.

Gudjonsson, K. (2015b). Using pinfo. https://github.com/log2timeline/plaso/wiki/Using-pinfo.

Hargreaves, C., & Patterson, J. (2012). An automated timeline reconstruction approach for digital forensic investigations. *Digital Investigation*, *9*, S69–S79.

Harrell, C. (2011). What's a timeline. http://journeyintoir.blogspot.com/2011/09/whats-timeline.html.

Ieong, R. S. (2006). Forza–digital forensics investigation framework that incorporate legal issues. *digital investigation*, *3*, 29–36.

Maurer, M. (2016). Evidence fetcher (efetch). https://github.com/maurermj08/efetch.

McQuaid, J. (2014). Forensic analysis of prefetch files in windows. https://www.magnetforensics.com/computer-forensics/forensic-analysis-of-prefetch-files-in-windows/.

Metz, J. (2015). Using psort. https://github.com/log2timeline/plaso/wiki/Using-psort.

Metz, J. (2017). Log2timeline/plaso. https://github.com/log2timeline/plaso/wiki.

Olsson, J., & Boldt, M. (2009). Computer forensic timeline visualization tool. *digital investigation*, *6*, S78–S87.

Quick, D., & Choo, K.-K. R. (2014). Impacts of increasing volume of digital forensic data: A survey and future research challenges. *Digital Investigation*, *11*, 273–294.

SANS Institutes (2011). Log2timeline cheatsheet. https://digital-forensics.sans.org/media/log2timeline_cheatsheet.pdf.

Walter, J. (2016). Kibana and SANS Evidence of ... . http://www.carpeindicium.com/blog/kibana-sans-evidence-of/.

Weber, R. (2017). How to use log2timeline! https://medium.com/dfclub/how-to-use-log2timeline-54377e24872a.

Wiedeman, G. (2016). Practical digital forensics at accession for born-digital institutional records. *Code4Lib Journal*, *31*.

Appendix A. Log2Timeline parsers

Below is a list of all available Log2Timeline / plaso parsers which we received by running the `log2timeline.py --help` in the Terminal.

```
***************************** Parsers *****************************
                Name : Description
--------------------------------------------------------------------
             amcache : Parser for Amcache Registry entries.
   android_app_usage : Parser for Android usage-history.xml files.
             asl_log : Parser for ASL log files.
                bash : Parser for Bash history files
             bencode : Parser for bencoded files.
      binary_cookies : Parser for Safari Binary Cookie files.
             bsm_log : Parser for BSM log files.
        chrome_cache : Parser for Chrome Cache files.
  chrome_preferences : Parser for Chrome Preferences files.
            cups_ipp : Parser for CUPS IPP files.
 custom_destinations : Parser for *.customDestinations-ms files.
          dockerjson : Parser for JSON Docker files.
                dpkg : Parser for Debian dpkg.log files.
               esedb : Parser for Extensible Storage Engine (ESE)
                       database files.
            filestat : Parser for file system stat information.
       firefox_cache : Parser for Firefox Cache version 1 files
                       (Firefox 31 or earlier).
      firefox_cache2 : Parser for Firefox Cache version 2 files
                       (Firefox 32 or later).
            fsevents : Parser for fseventsd files.
            java_idx : Parser for Java WebStart Cache IDX files.
                 lnk : Parser for Windows Shortcut (LNK) files.
 mac_appfirewall_log : Parser for appfirewall.log files.
        mac_keychain : Parser for MacOS Keychain files.
       mac_securityd : Parser for MacOS securityd log files.
             mactime : Parser for SleuthKit version 3 bodyfiles.
             macwifi : Parser for MacOS wifi.log files.
   mcafee_protection : Parser for McAfee AV Access Protection log
                       files.
                 mft : Parser for NTFS $MFT metadata files.
              msiecf : Parser for MSIE Cache Files (MSIECF) also
                       known as index.dat.
               olecf : Parser for OLE Compound Files (OLECF).
             openxml : Parser for OpenXML (OXML) files.
        opera_global : Parser for Opera global_history.dat files.
 opera_typed_history : Parser for Opera typed_history.xml files.
                  pe : Parser for Portable Executable (PE) files.
               plist : Parser for binary and text plist files.
          pls_recall : Parser for PL/SQL Recall files.
  popularity_contest : Parser for popularity contest log files.
            prefetch : Parser for Windows Prefetch files.
         recycle_bin : Parser for Windows $Recycle.Bin $I files.
   recycle_bin_info2 : Parser for Windows Recycler INFO2 files.
               rplog : Parser for Windows Restore Point (rp.log)
                       files.
                sccm : Parser for SCCM logs files.
             selinux : Parser for SELinux audit.log files.
        skydrive_log : Parser for OneDrive (or SkyDrive) log
                       files.
    skydrive_log_old : Parser for OneDrive (or SkyDrive) old log
                       files.
           sophos_av : Parser for Anti-Virus log (SAV.txt) files.
              sqlite : Parser for SQLite database files.
    symantec_scanlog : Parser for Symantec Anti-Virus log files.
              syslog : Syslog Parser
             usnjrnl : Parser for NTFS USN change journal
                       ($UsnJrnl).
                utmp : Parser for Linux/Unix UTMP files.
               utmpx : Parser for UTMPX files.
              winevt : Parser for Windows EventLog (EVT) files.
             winevtx : Parser for Windows XML EventLog (EVTX) files.
         winfirewall : Parser for Windows Firewall Log files.
              winiis : Parser for Microsoft IIS log files.
              winjob : Parser for Windows Scheduled Task job
                      (or At-job) files.
              winreg : Parser for Windows NT Registry (REGF) files.
            xchatlog : Parser for XChat log files.
     xchatscrollback : Parser for XChat scrollback log files.
zsh_extended_history : Parser for ZSH extended history files
--------------------------------------------------------------------
```

Appendix B. Log2Timline output modules

As mentioned in Sec. 2.2, the framework comes with a tool psort.py that allows filtering, sorting and converting the plaso storage file in one of the following output formats which is slightlty different to the list releases on github Metz (2015):

```
$ psort.py -o list

************************* Output Modules *************************
          Name : Description
------------------------------------------------------------------
        l2tcsv : CSV format used by legacy log2timeline, with 17
                 fixed fields.
          xlsx : Excel Spreadsheet (XLSX) output
        l2ttln : Extended TLN 7 field | delimited output.
4n6time_sqlite : Saves the data in a SQLite database, used by the
                 tool 4n6time.
 4n6time_mysql : MySQL database output for the 4n6time tool.
           kml : Saves events with geography data into a KML format.
       dynamic : Dynamic selection of fields for a separated value
                 output format.
         rawpy : "raw" (or native) Python output.
          json : Saves the events into a JSON format.
          null : Output module that does not output anything.
    timesketch : Create a Timesketch timeline.
           tln : TLN 5 field | delimited output.
     json_line : Saves the events into a JSON line format.
       elastic : Saves the events into an Elasticsearch database.
------------------------------------------------------------------
```