DFRWS

An automated timeline reconstruction approach for digital forensic investigations

Christopher Hargreaves[a]* and Jonathan Patterson[a]

[a]*Centre for Forensic Computing, Cranfield University, SN6 8LA, UK*

Abstract

Existing work on digital forensics timeline generation focuses on extracting times from a disk image into a timeline. Such an approach can produce several million 'low-level' events (e.g. a file modification or a Registry key update) for a single disk. This paper proposes a technique that can automatically reconstruct high-level events (e.g. connection of a USB stick) from this set of low-level events. The paper describes a framework that extracts low-level events to a *SQLite* backing store which is automatically analysed for patterns. The provenance of any high-level events is also preserved, meaning that from a high-level event it is possible to determine the low-level events that caused its inference, and from those, the raw data that caused the low-level event to be initially created can also be viewed. The paper also shows how such high-level events can be visualised using existing tools.

Keywords: digital forensics; automation; timelines; event reconstruction;

1. Introduction

This paper presents an automated approach to assist analysts during a digital investigation by automatically reconstructing high-level events that have occurred, for example, the connection of a USB stick. The paper makes the following contributions: a *Python* based prototype that extracts dates and times from various files on a mounted disk image; a framework that allows 'analysers' to be written that can produce a high-level event based on the presence of one or more low-level events. This combined approach also allows detailed provenance of any automatic inferences to be preserved, from a high-level event, through the low-level events that were (and were not) present, all the way down to the raw data that caused a low-level event to be extracted in the first place.

The remainder of this paper is structured as follows: Section 2 provides background information and justifies a timeline-based approach in a digital investigation. Section 3 considers related work, and Section 4 discusses the methodology for this research. Section 5 presents the design and implementation. Section 6 provides sample results from the use of the developed tool and Sections 7 and 8 evaluate the research and discuss conclusions and further work.

2. Background

One of the challenges to digital forensics is the volume of data that needs to be analysed. This has arisen as a result of a number of factors, including the importance of digital evidence in a broader range of investigations, increasing storage capacities, and the increasing number of digital devices owned by an individual [1]. As a result, automation is an increasingly important part of digital forensics. Many existing automated tools have focused on the extraction stage of a digital investigation, i.e. making more information accessible from the raw data, and are very effective at this. For example, *Internet Evidence Finder* automates the recovery of artefacts on a disk image that relate to certain Internet use, e.g. *Facebook* chat artefacts.

Automated extraction approaches do help with digital investigations, but only in that they make more data available to an analyst in a form that can be understood. Since it is usually necessary to answer questions about previous digital events [2], one approach that can help is the use of timelines. Existing work on timelines in digital investigations is discussed in the following section and the timeline generation techniques are divided into 'file system only' and 'enhanced'.

* Corresponding author. Tel.: +44 (0)1793 785993.
E-mail address: c.j.hargreaves@cranfield.ac.uk.

3. Related work

3.1. Timelines based on file system times

The majority of commercially available forensic software packages reconstruct the file system of a disk image and make the contents of files and their metadata accessible. Depending on the file system in use, this metadata usually includes at least Modified, Accessed and Created (MAC) times. Some forensic tools are capable of turning the multiple times associated with a file into a timeline. For example Carrier describes how to generate a file activity timeline using *The Sleuth Kit* [3], and Bunting describes the graphical 'Timeline View' of *EnCase* [4]. The limitation of file system metadata based timelines is that they do not consider times that are available by examining the contents of files.

3.2. Timelines including times from inside files

Olsson and Boldt improved upon file metadata based timelines with the *Cyber Forensic Time Lab (CFTL)* [5]. This tool not only recovers file system times from FAT and NTFS volumes, but also extracts times from a variety of files, for example EXIF data, Link files, MBOX Archives and Windows Registry files. Interestingly, *CFTL* also maintains some information about the source of extracted events. It is also suggested that an extension of the work could be to automatically search for "certain predefined patterns of suspicious activity, helping the investigator to spot interesting parts of the timeline more efficiently".

Also, *log2timeline* [6], with the *timescanner* enhancement can automatically and recursively examine files and directories. If an appropriate 'input module' is available for a file, times are extracted and added to a timeline. Reference [6] also hints at the possibility of grouping events that are part of the same activity when describing the potential future use of the 'super event' table in the *SQLite* output format.

A more detailed review of available timeline software is available in [7], but the examples in this sub-section demonstrate that there are a number of benefits to using an 'enhanced' timeline in addition to improving the richness of the timeline, i.e. increasing the number of events. As discussed in [5], a tool such as *Timestomp* could be used to clear file system times, but this would not affect times within files. Even if not overwritten maliciously, file access times can be updated in bulk by anti-virus products [6] or the updating of them disabled by default in modern operating systems or by altering a Registry key.

3.3. Visualisations

There is also some work that discusses the visualisation of digital forensic timelines. For example, *EnCase*'s visualisation is mentioned in Section 3.1 [4].

Buchholz and Falk [8] developed *Zeitline,* which is a GUI based tool that allows file system times to be imported from *The Sleuth Kit* and other sources (using Import Filters). This tool provides searching and filtering of events. It also introduces the concepts of *atomic events* and *complex events,* where the former are "events that are directly imported from the system" and the latter are "comprised of atomic events or other complex events". *Zeitline* allows an investigator to manually combine atomic events into complex events.

Aftertime [9] is a Java based application that not only performs enhanced timeline generation (as described in Section 3.2) from a disk image, but also visualises the results as a histogram, with time on the x-axis against numbers of different events on the y-axis.

Lowman discusses several visualisations of web history data, including heat maps, bar charts of activity, word clouds and a timeline view [10]. The results presented suggest the visualisations are effective. However, it is difficult to know how they would scale if they included all events from a disk, as described in the previous section.

One of the problems that all visualisations will face is that when all the times are extracted from file system metadata and from within files, hundreds of thousands of events can be produced. Finding ways to display this number of events in a way that is useful to an investigator is extremely challenging; as [6] states, "a super timeline often contains too many events for the investigator to fully analyze, making data reduction or an easier method of examining the timeline essential".

3.4. Summary

This section has shown the importance of recovering times from inside files in addition to using file system metadata. However, this causes new problems, as the large number of events produced are difficult to analyse and extremely problematic to visualise in a manner that is useful. It has been suggested that highlighting certain patterns of activity to indicate areas of interest in the timeline may be an effective approach, and that it is important to maintain records of the source of extracted data.

4. Methodology

Many of the problems highlighted in the related work section stem from the volume of data created when all times are extracted from a disk image, particularly with the 'super-timeline' approach. This volume of data, or in particular, the number of events that are generated, makes analysis difficult and limits the way in which data can be visualised.

This research aims to determine the extent to which it is possible to automate the manual process that an investigator can undertake to combine multiple 'low-level' events, (i.e. data extracted from file systems and compound files) into 'high-level', human-understandable events, e.g. connection of a USB stick. Such an approach would produce a summary of activity that would assist in focusing an investigation on particular areas of interest or perhaps prioritising the investigation of one machine over another.

The chosen research method in this case is the development of a software prototype. This is chosen over a design-based approach, as it means it is also necessary to overcome any practical issues that are difficult to identify at the design stage.

Another methodological question that arises is whether to build on top of existing frameworks, e.g. *log2timeline*. In the literature review there are several identified properties that are desirable in a timeline reconstruction system. One of which is traceability of any results back to the original low-level data [5]. If *log2timeline* were to be used, the input modules would need to be enhanced to accommodate this. To allow for such updating, the data structures that they process would need to be understood to such an extent that code could be implemented entirely anyway. As a result, this work implements a complete framework for low-level event extraction that does replicate much of the *log2timeline* capability, but with some enhancements. Since this is a fresh implementation then it is not necessary to implement in any particular language based on legacy requirements. The implementation language in this case is Python 3, which can be considered to be very suited to digital forensics as a result of its emphasis on readability of code, which means that inspection of methods used can be more easily achieved by a third-party.

5. Design

This section discusses the design of the timeline reconstruction framework. It provides an overview, followed by details of the low-level event generation and the high-level event reconstruction.

5.1. Overall design

The developed software (*Python Digital Forensic Timeline (PyDFT)*) functions in two main stages: low-level event extraction and high-level event reconstruction. There are many other supporting components that allow this to occur, including case management, conversion of different formats for storing date-times, and some basic GUIs. However, due to space constraints these will not be discussed in detail and the focus will be on the generation of low-level events and the analysis of these low-level events to produce high-level events. The sections that follow will discuss the design and implementation of these two stages.

5.2. Generation of low-level events

5.2.1. Overview

The generation of low-level events includes both file system times, and times extracted from inside files. Like *log2timeline*, this is achieved by the analysis of a mounted file system, rather than the disk image based approach of *Aftertime*. While most forensic investigations will be working from a full disk image, an equivalent mounted file system can be easily obtained by mounting the disk image in read-only mode using both *Mac OS X* and *Linux*.

5.2.2. Extraction of file system times

Since a mounted file system is being examined, it is possible to obtain file system times using OS level commands. However, this provides a limited set of times, e.g. file creation times are not available on all systems. Therefore, for NTFS file systems, the times are not obtained by querying the file system, but instead a check is performed for the presence of the $MFT file, that for NTFS, contains all the data necessary to extract low-level events about the file system. This can be accessed directly on *Linux,* and on *Mac OS X* using an *NTFS* driver from *Tuxera*. If the $MFT file is found, it is processed and the modified, accessed, created and entry modified times from the Standard Information Attribute are used to build four events for each file on the file system.

5.2.3. Times from inside files

After the generation of the low-level events using the MFT, as discussed in the related work section, there are a significant number of dates and times that can be extracted from inside complex files such as Windows Registry hives. This is handled by an 'Extractor Manager'. For each file in the mounted file system, the function GetTimesFromInsideFiles() is called and the file is checked to see if any of the time extractors available can be used for that file. In order to determine which (if any) event extractor can be used for a particular file, the file name, file path and a file pointer (which can be use to read bytes in order to perform signature based matching) are available. Upon identifying a file that can be processed further, the appropriate time extractor is called, events are generated from the times within the file, and these are added to the low-level timeline. Time extractors that have currently been implemented include: *Chrome*, *Firefox* and *Internet Explorer* history, *Skype*, *Windows Live Mail*, *XP* Event Logs, Link Files, Registry, and Setup API.

5.2.4. Parsers and bridges

Each extractor is made up of two parts: *parsers*, which process the raw data structures and recover data in a usable form; and *bridges*, which take the information that a parser provides and maps the values to a low-level event object. This design makes it easier to accommodate new parsers when they are developed for data structures (as part of other research). It also makes the code in the parsers reusable for other research, since they are not dependant on components and data structures of the Python Digital Forensic Timeline.

5.2.5. Traceability

Since these time extractors have been re-implemented specifically for the purpose of timeline generation and analysis, this has allowed the concept of traceability to be built in at every layer. Practically this means that when a low-level event is returned from a file by an extractor, it also returns a value that allows easy access to the raw data from which this event was produced. However, not all data is retrieved in exactly the same way and there are currently several types of provenance that can be returned with the low-level event.

offset: A byte offset within a file, e.g. Windows Registry, index.dats.
line number: A line number within a text based file e.g. Setupapi.log.
SQL Query and record ID: Used when the source file is a database and the row number of the results of a specific query can be used, e.g. Firefox history.
third party: In addition to the extractors developed as part of this research, it may be desirable to use third-party code, and while that code may not be designed to preserve provenance to the level of file offsets, this at least allows the third-party plugin to be recorded.

Whichever type of provenance is used, the file from which the data was extracted is always captured within the low-level event.

5.2.6. Low-level event format

As discussed in the previous sub-section, one of the details preserved in the low-level event format is the provenance, i.e. the data that is the source of the low-level event. However, there are a number of other fields that make up a low-level event. These are discussed below.

id: A unique identifier for each event.
date_time_min: The earliest time that the event could have occurred.
date_time_max: The latest time that the event could have occurred.
evidence: The evidence item that the event came from. Since the case being processed may contain multiple sources of evidence, this allows these multiple sources to be incorporated into a single timeline. This can be useful in cases where multiple computers or devices are involved, or even the analysis of a computer system with multiple hard disks.
plugin: The time extractor that was used to recover the event, e.g. 'Registry'.
type: The type of the event, e.g. File Created, Key Last Updated, URL Last Visited. This is currently determined by the author of an extractor.
path: The object that the event relates to, e.g. a file or URL.
provenance: The data that was used to produce the event. As previously discussed this includes the file that the event came from, and any further details that assist in directly accessing the relevant raw data, e.g. the offset within the file.
keys: Optional, additional details about an event.

The fields stored are similar to those used in existing tools such as *Zeitline*, *log2timeline* and *Aftertime.* The specific details of what is stored and the reasoning for some of the differences from existing formats are discussed in the following paragraphs.

Event ID: The event ID uniquely identifies each low-level event. This is necessary for the later high-level event reconstruction as will be discussed in Section 5.3.

Maximum and Minimum date/times: This allows times that are stored on disk imprecisely to still be used to generate low-level events, for example file access times on FAT file systems record only the date of the access not the precise time. Storing a maximum and minimum is more technically accurate than simply rounding the access time to midnight, since in reality there is a 24-hour period in which that access could have taken place. Dates and times are stored in UTC as Unix Time (seconds since 1st January 1970). However, fractional numbers are permitted, meaning that precision is not lost when times are recovered from files that record times with more detail than Unix Times, e.g. Windows FILETIME (100-nanosecond intervals since 1st January 1601).

Keys: The keys field is implemented as a *Python* dictionary. It is used to store any further relevant information that does not fall under the fields outlined previously. The data stored will depend on the type of the event. File related events may capture the *file size* and *user ID*, whereas URL accesses may contain *page title* and *visit type.* This is similar to the *log2timeline* concept of 'extras'.

As discussed above, many of these fields can be mapped directly to other low-level timeline generators such as those detailed in Section 3. This is a useful property as it should be relatively simple to allow low-level events to be imported from other tools. However, such imported events would have the limitation of not preserving as much provenance information, therefore not allowing full traceability back to the raw data.

5.2.7. Backing store for the low-level timeline

While internally in *PyDFT*, low-level events are implemented as a Python class, it is also necessary to have a backing store for the low-level timeline. It is worth noting that early versions of the low-level timeline generation tool kept the data in memory only, but the timeline quickly became too large, causing disk paging and resulting in the associated performance problems. Existing low-level timeline tools have implemented a variety of different means of storing timelines. It is common to export the generated timelines to disk in CSV form. This has the advantage of allowing commands such as *grep* to be used to query for specific events [6].

However, this tool uses *SQLite* as a backing store since it is necessary to conduct multiple advanced queries on the data set and it has been found to offer performance benefits to using a flat file structure (although *PyDFT* can also export low-level timelines to several other formats). *SQLite* is discussed as a backing store option in [7], but is described as having the limitation of requiring an investigator to have knowledge of SQL in order to search for particular events. As will be seen in Section 5.3, in this implementation, this is not the case, and therefore not a limitation.

The *SQLite* database in this implementation comprises three tables; *info*, *events* and *keydata*. The *info* table contains metadata such as the version of the timeline tool that produced that database and the time of its creation. The majority of data is stored in the *events* table, with data from the *keys* field being stored in the separate *keydata* table. This structure results in some duplication of data, but as the database is not subject to updates, this is not problematic.

5.2.8. Summary

The low-level events are extracted from inside files using an 'extractor manager'. Events are converted into a standard format for a low-level event and added to a timeline. This timeline is stored as a *SQLite* database which can be used for further queries. In addition to fields such as dates and times of the event, the event type, and path to which the event relates, the provenance of the data is also stored, which can contain details such as the offset of the raw data from which the event was generated.

5.3. Reconstruction of high-level events

5.3.1. Overview

The previous section described how a low-level timeline is generated and events added to a *SQLite* database. This section discusses how this timeline is automatically processed to produce high-level, human-understandable events.

Some previous work has proposed neural networks for automated event reconstruction [11]. However, the approach in this paper searches for patterns of events in the low-level timeline based on pre-determined rules. The approach is based on a plugin framework where each plugin is a script that detects a particular type of high-level event. Each 'analyser' script contains criteria that specify the low-level events that should be present if that high-level event occurred, and searches the entire timeline for low-level events that match within a specified period of time. An analyser is made up of a number of components, which are discussed in the following sub-sections.

5.3.2. Basic event matching using test events

One of the limitations of using a *SQLite* database to store low-level events discussed in [7] was that knowledge of SQL is necessary to query the database. It is not certain that this is a limitation, but in any case, SQL knowledge is not necessary in this implementation due to the use of 'Test Events'. This means that in order to find a low-level event in the timeline it is simply necessary to construct a low-level event that has the properties of the one that needs to be matched. A method can then be called on an event in the low-level timeline to determine if it matches the 'test event' that has been constructed. During execution of the matching method, each of the fields of a low-level event are compared, but exact matching is not necessary. The matching is implemented in such a way that regular expressions can be specified in any of the fields in a test event. If a field is not specified then it is not evaluated as part of the match. This allows test events to be constructed as shown below, which matches any file creation on the file system that has the extension *.doc* or *.docx*. Dates and times are not compared during matching since it is desirable to match events anywhere in the timeline.

```
test_event = PyDFT.Core.LowLevelEvent.LowLevelEvent()
test_event.provenance_source = "File System"
test_event.type = "Created"
test_event.path = "\.docx?$"
```

The remainder of this simple analyser searches the entire timeline for events that match, builds appropriate high-level events, and adds them to the high-level timeline.

```
for each_event in timeline:
    if each_event.Match(test_event):
        # Create high-level event
        # Add high-level event to high-level timeline
```

5.3.3. Matching multiple artefacts

In addition to building a high-level event from a single low-level event, it is possible, and often preferable to use multiple low-level events. This approach means that one or more 'test events' are constructed and these act as triggers. The timeline is searched as before and if any of the test events are matched then a working hypothesis that a particular high-level event occurred is created. At this point a new low-level timeline is created in memory by sub-sampling the timeline within a period defined in the analyser, for example 10 seconds either side of the trigger event. This timeline is then searched for all of the low-level events that would expect to be seen for this high-level event. Events that are matched within this period are added to a list of supporting artefacts (similar to *complex* events consisting of *atomic* events in [8]). If they are not found then they are added to a list of contradictory artefacts. This approach can be thought of as 'temporal proximity pattern matching' for low-level digital events.

Whether a high-level event is created or not based on a trigger alone depends on the specific analyser. Some may create a high-level event with only one low-level event matched, whereas others may require several low-level events to be present.

This 'reasoning' is stored within the high-level event and there are three additional fields, (trigger, supporting and contradictory). Each of the three fields store: 'Reasoning Artefacts' which contain: the ID of the low-level event that matched, a description of the reason, and the test event that caused the match.

There are different ways in which analysers that use multiple artefacts may be constructed. Some specify a static set of test events at the start of the analyser. For example, a 'Firefox Installation' analyser creates two test events, one for creation of a *Firefox* executable and another for the creation of a *Mozilla Firefox* folder in Program Files. Other, more complex analysers dynamically create new test events based on data extracted from the trigger event. For example, a 'USB Connection' analyser first searches for an entry in the setupapi.log, then builds an additional test event using the serial number of the USB device extracted from the trigger event. This means that only low-level events related to the trigger are matched. This is shown in Section 6.2.

Currently, there are 22 analysers implemented. Some examples of which include: 'User Creation', 'Windows Installation', 'Google Search', 'YouTube Video Access', 'Skype Call' and 'USB Connected'.

5.3.4. High-level event format

The structure of a high-level event is similar to that of a low-level event and some fields are inherited from the same superclass.

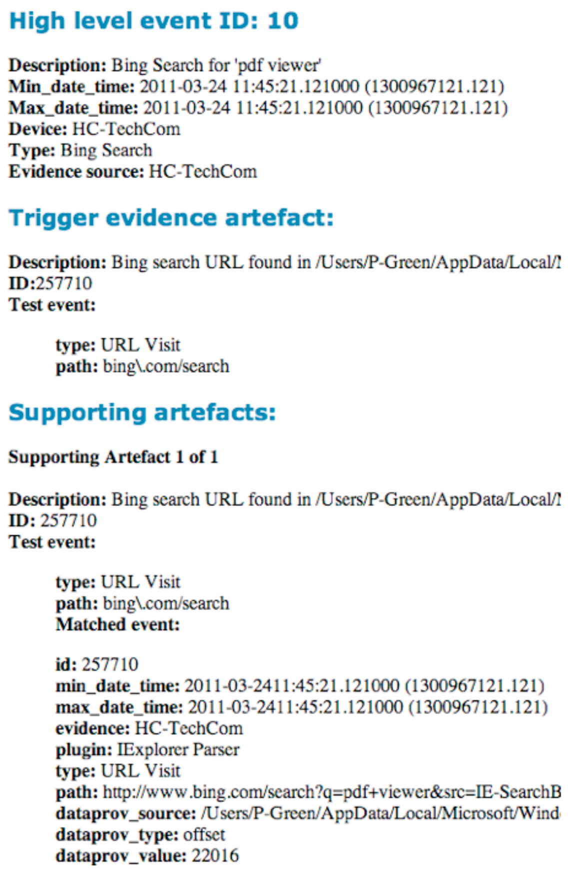
id: A unique identifier for each high-level event.
date_time_min: The earliest time that the event could have occurred.
date_time_max: The latest time that the event could have occurred.
evidence_source: The evidence item from which the event came.
type: The type of the event, e.g. 'USB Connected' or 'Program Installed'.
description: A human readable description of the event.
category: Since many analysers may be run, a category can be applied to each event for easy filtering.
device: The device that the event occurred on. For example, a photograph may be stored on a computer and contains EXIF data indicating it was taken with an *iPhone*. If the event being reconstructed was 'Photograph Taken', then the 'device' would be the

iPhone, but the ‘evidence_source’ would be the computer. This is interesting as it can produce timelines for devices that are not necessarily in the possession of the investigator.
summary: This is not yet implemented but allows for short summaries of documents, websites etc. to be included in the event. This may assist in future searching and filtering of events.
files: When the high-level timeline is generated, files that are associated with an event are copied to a folder within the case path for previewing or further analysis. This field contains multiple entries and maintains an index of copied files.
keys: This field allows additional details about an event to be captured, e.g. USB Connection events store VIDs, PIDs and Serials.
trigger_evidence_artefact*:* This stores the reasoning artefact for the low-level event that triggered the search for additional low-level events.
supporting_evidence_artefacts*:* This is a list and stores all the additional reasoning artefacts that were found.
contradictory_evidence_artefacts: This is also a list and stores all the additional reasoning artefacts that were searched for but not found within the timeframe.

5.3.5. High-timeline output

The main output from the tool is currently XML, since it has not yet been necessary to move the high-level timeline into *SQLite*. Example XML output for an event is shown later. However, in addition to the XML, other formats are also exported in order to take advantage of some existing visualisation software. Examples are shown later in Section 6.

In addition to the representations of the entire high-level timeline that are saved, optionally, individual HTML reports are also created for each high-level event. This allows more detail to be displayed since the references to low-level events stored in high-level events are simply IDs. In the HTML reports of individual high-level events, the low-level event ID is accessed and the event’s full details displayed. This is shown in Figure 1. Also, optionally included in the HTML report are any other low-level events that occur within a set time period around the high-level event, which may assist with ‘temporal proximity’ analysis as described in [6].



High level event ID: 10

Description: Bing Search for 'pdf viewer'
Min_date_time: 2011-03-24 11:45:21.121000 (1300967121.121)
Max_date_time: 2011-03-24 11:45:21.121000 (1300967121.121)
Device: HC-TechCom
Type: Bing Search
Evidence source: HC-TechCom

Trigger evidence artefact:

Description: Bing search URL found in /Users/P-Green/AppData/Local/
ID:257710
Test event:

  type: URL Visit
  path: bing\.com/search

Supporting artefacts:

Supporting Artefact 1 of 1

Description: Bing search URL found in /Users/P-Green/AppData/Local/
ID: 257710
Test event:

  type: URL Visit
  path: bing\.com/search
  Matched event:

  id: 257710
  min_date_time: 2011-03-2411:45:21.121000 (1300967121.121)
  max_date_time: 2011-03-2411:45:21.121000 (1300967121.121)
  evidence: HC-TechCom
  plugin: IExplorer Parser
  type: URL Visit
  path: http://www.bing.com/search?q=pdf+viewer&src=IE-SearchB
  dataprov_source: /Users/P-Green/AppData/Local/Microsoft/Wind
  dataprov_type: offset
  dataprov_value: 22016

Figure 1: HTML event report of a Bing search showing trigger and supporting artefacts.

Other output includes a full log of all actions performed during the timeline generation and analysis, including any errors or warnings.

5.3.6. Summary

High-level events can be created by scanning the complete timeline for low-level events that match specific criteria. This is achieved by creating test events that are low-level events with the properties that it is desirable to match. The ‘reasoning’ of the high-level event is preserved since the low-level events that were matched, and any test events that did not match are recorded within the high-level event. Since the IDs of the low-level events are captured, it is then possible to use this to obtain the low-level event provenance, which ultimately links back to the original data in the disk image, e.g. a file and an offset within it.

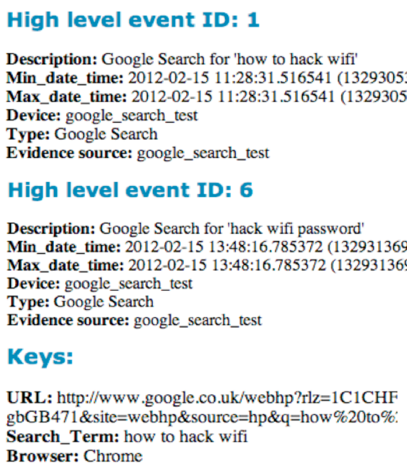
6. Results

This section demonstrates a small sample of the high-level events that can be reconstructed using the developed system.

6.1. Google searches

This simple example illustrates the detection of Google searches. Several searches were conducted on a test system and the times noted. The notes recorded were:
- 11:28:30 Google search for 'how to hack wifi',
- 13:48:15 Google search for 'hack wifi password'

The corresponding HTML reports from PyDFT are shown in Figure 2.



High level event ID: 1

Description: Google Search for 'how to hack wifi'
Min_date_time: 2012-02-15 11:28:31.516541 (1329305311.5165405)
Max_date_time: 2012-02-15 11:28:31.516541 (1329305311.5165405)
Device: google_search_test
Type: Google Search
Evidence source: google_search_test

High level event ID: 6

Description: Google Search for 'hack wifi password'
Min_date_time: 2012-02-15 13:48:16.785372 (1329313696.7853718)
Max_date_time: 2012-02-15 13:48:16.785372 (1329313696.7853718)
Device: google_search_test
Type: Google Search
Evidence source: google_search_test

Keys:

URL: http://www.google.co.uk/webhp?rlz=1C1CHF gbGB471&site=webhp&source=hp&q=how%20to%
Search_Term: how to hack wifi
Browser: Chrome

Figure 2. The HTML reports for the two Google searches

In this case a specific low-level event is directly mapped to a high-level one, but this still succeeds in filtering the data to that which you are interested in.

6.2. USB device connection

This example shows that the connection of a USB device that occurred at 13:52:45 is automatically detected and there are several supporting artefacts. Figure 3 shows an extract of the high-level event as XML. The time 1329313992 converts to 13:53:12, which is shortly after the actual connection.

```
<event>
    <event_id>30</event_id>
    <min_date_time>1329313992</min_date_time>
    <max_date_time>1329313992</max_date_time>
    <evidence_source>test</evidence_source>
    <description>USB device connected (E:/)</description>
    <category>User</category>
    <device>test</device>
    <type>USB connected</type>
    <files></files>
    <summary></summary>
    <key name="VID">07AB</key>
    <key name="PID">FCF6</key>
    <key name="Rev">5.00</key>
    <key name="Disk&Ven"></key>
    <key name="FriendlyName">E:\</key>
    <key name="Serial">07A80207B128BE08</key>
    <key name="Prod">Freecom_Databar</key>
```

Figure 3. The XML representation of a 'USB Connection' high-level event

The supporting artefacts that were identified for this event are described below but not shown as XML due to space constraints. The first is the trigger event, and those that follow were matched from test events dynamically generated to contain the serial number extracted from the setupapi log.

- "Setup API entry for USB found (VID:07AB PID:FCF6 Serial:07A80207B128BE08)"
- "Setup API USBSTOR entry found"
- "USBStor details found in Registry"
- "Windows Portable Device entry found in Registry"

6.3. Visualisation

Since the number of high-level events is significantly smaller than the number of low-level events, it is possible to use existing visualisation tools to effectively visualise the history of a computer system. Figure 4 shows the visualisation software *TimeFlow* being used to visualise the automatically reconstructed high-level events.

In the time period of the sequence of high-level events shown in Figure 4 (which takes place over approximately 5 minutes), in the equivalent low-level timeline, there are 2,894 low-level events during this time. This would be difficult to visualise in a useful manner, particularly when this is scaled up to the entire history of a disk.

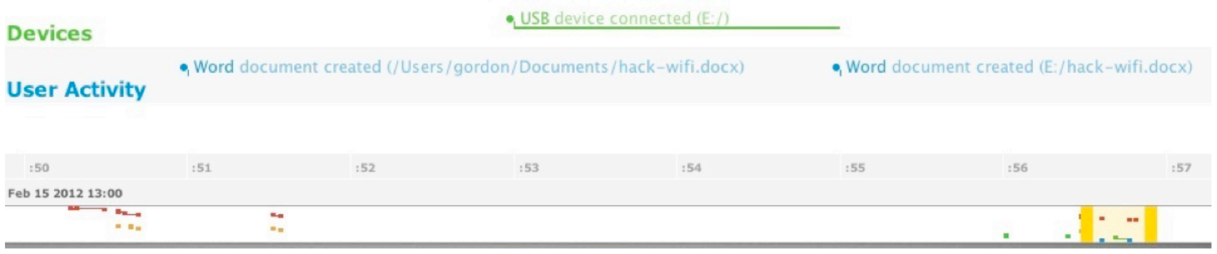

USB device connected (E:/)
Devices
Word document created (/Users/gordon/Documents/hack-wifi.docx)
Word document created (E:/hack-wifi.docx)
User Activity
:50
:51
:52
:53
:54
:55
:56
:57
Feb 15 2012 13:00

Figure 4. *Timeflow* visualisation of three events: creation of hack-wifi.docx on local drive, connection of USB, and creation of hack-wifi.docx on E: drive

6.4. Performance

While many factors can affect the time taken for timeline generation and analysis, some example figures are provided in Table 1 to show that the time taken is in the region of what is acceptable for digital forensic tools, which are often left to run overnight for keyword indexing, searching or hashing. All of these numbers are produced on between 2.2 and 2.8GHz Core 2 Duo machines, with 4-8GB of RAM. It should be noted that at this stage no attempts have been made to optimise code for performance, or take advantage of multiple CPUs.

Table 1. Example times for timeline generation and analysis. The first is from a small, test VM, others are from 'real world' systems.

Volume size	Approx. time system in use	Low-events produced	Time for low generation (hh:mm)
20GB	2 months	0.6 million	0:15
100GB	2 years	1.2 million	0:42
250GB	5 years	1.6 million	1:05
Volume size	Number of analysers	High-events produced	Time for analysis
20GB	19	666	0:28
100GB	19	2704	1:10
250GB	17	3902	1:14

Based on all performance data collected so far, the timeline analysis takes approximately 2 minutes per analyser, per 1 million events.

7. Evaluation

The results section has shown that a 'temporal proximity pattern matching' approach is feasible as an automated event reconstruction technique. Whilst results have shown it may be feasible, many more 'analysers' need to be written to further test this hypothesis. Development of more 'analysers' and 'time extractors' will also further test the appropriateness of the low-level and high-level event formats that are currently in use. However, the flexible 'keys' field has accommodated all new low and high-level events so far. In addition, while not presented in full here, use of the maximum and minimum times for high-level events are extremely useful and will be discussed in detail in a future paper. They may also provide flexibility for more advanced event reconstruction, such as techniques that use restore points to determine that an event occurred between two dates [12].

Regarding low-level time extractors, it should be noted that there are currently some gaps in the scope of extraction of dates and times from a disk. For example, Windows Vista/7 Event Logs, Recycle Bin and Prefetch time-extractors have not yet been implemented. However, since the developed system is plugin based, this is simply due to time restrictions, rather than any technical barrier.

It has also been shown that preserving the provenance of inferred high-level events is possible, including the low-level events that support the inference, as well as the location of the raw data that was initially used to create the low-level event. However, it is difficult to evidence the assertion that this is a desirable property without studies of analysts investigating scenarios using multiple methods.

The performance of the prototype has also been demonstrated, and this is considered to be within acceptable timeframes for forensic analysis tools. However, there is currently a possible performance bottleneck since each additional analyser searches the entire timeline linearly looking for patterns. Adding new analysers therefore increases the time taken. However, there are many opportunities for optimisation, e.g. the prototype has not yet been adapted for multi-core systems. Work is also underway investigating whether further secondary indexing of the *SQLite* database can improve performance of searching the timeline, therefore reducing this impact.

A fundamental assumption is that the clock on the system being investigated is correct. This assumption has two main problems: one is that it is not currently possible to apply a generic clock offset (i.e. after checking the BIOS time of the machine) which would be applied to all times in the timeline. However, this is a fairly minor update to the *LowLevelTimeline* class that will soon be implemented. Secondly, there is no mechanism for detecting deliberate manipulation of the clock. However, whilst this is a related topic, it is believed to be a separate research area, and there are several examples of previous work on the subject.

An area that requires significant development is testing of this prototype. While many individual time extractors have been developed using Test Driven Development and are quite robust, holistic validation of the accuracy of the low-level timeline is still in progress, as is validation of the accuracy of the high-level timeline output. The former can be partially addressed by comparing existing timeline generation tools with the low-level output, assuming similar *time extractors*/*input modules* are selected. This may be an interesting area to explore for both tools, since they have been developed independently and may offer real dual-tool verification. The latter (high-timeline output) can be tested either through comparison with reality (thorough documentation of actions performed to a test system), or by comparing the automatically inferred results to those from a traditional, full forensic examination. It is deemed likely that a combination of these approaches will offer the most thorough testing strategy.

8. Future work

Initial future work involves increasing the number of analysers in order to further test the hypothesis that such 'temporal proximity pattern matching' is effective. This may also require additional low-level event extractors to be developed, as there are some notable exceptions. This development may also include formalising the types of low-level event that can be used. It may also be interesting to explore inputting data from other tools into the timeline, although as explained earlier, this introduces limitations to the full provenance of reconstructed events.

It is also necessary to continue testing the framework against 'real world' data (some examples of which were included in Section 6.4), since these more extensively test the robustness and scalability of the framework than small data sets developed for testing individual extractors and analysers.

The complexity of the analysis scripts can also be explored further. With a framework in place that allows low-level events to be easily queried simply by building a low-level event with the required properties, it may be possible to investigate more advanced inference methods. For example, using Bayesian networks to attribute probabilities to different low-level events that need to be present in order to infer that a high-level event occurred, or considering how the order in which low-level events occur affects high-level event reconstruction [13].

Perhaps most interestingly, moving from hundreds of thousands of low-level events to a few hundred, human understandable, high-level events may open up new possibilities for visualisation of data from digital forensic investigations and enable the development of tools with much greater analysis capabilities.

9. Conclusions

This paper has shown that it is possible to use pattern matching to automatically reconstruct high-level, human-understandable events. It has also shown that using such high-level events makes useful visualisations much more feasible. However, the importance of maintaining details of how any inference is performed should be re-iterated, preferably with provenance that links back to the raw data from which low-level events were obtained.

It must also be stressed that this sort of automated approach is not intended as a replacement for a full forensic analysis by an experienced, trained analyst. It is hoped that with further development, such a technique could be integrated into the digital investigation process and help speed up analysis by performing pre-processing of disks while they are waiting to be analysed. Using such pre-processing, when an analyst receives a disk image for examination, it could be accompanied by a summary of the automated analysis, with areas of potential interest already highlighted, along with sufficient information such that all the results could be quickly verified against the raw data. This would allow more time for advanced analysis of areas of the disk that are non-trivial to examine. Such an approach may also be of particular use in cases where a large number of machines have been seized and it is necessary to identify disks that need to be prioritised for examination over others.

References

1. Turner, P., 2005. Unification of digital evidence from disparate sources. Digital Investigation, 2(3), pp.223–228.

2. Carrier, B., 2006. Risks of Live Digital Forensic Analysis. Communications of the ACM, 49(2), pp.56–61.

3. Carrier, B., 2003. File Activity Timelines. http://bandwidthco.com/whitepapers/compforensics/fsanalysis/File%20Activity%20Timelines.pdf.

4. Bunting, 2008, EnCE Study Guide, p.235-237

5. Olsson, J. & Boldt, M., 2009. Computer forensic timeline visualization tool. Digital Investigation, 6(S1), pp.S78–S87.

6. Guðjónsson, K., 2010. Mastering the Super Timeline with log2timeline. SANS Reading Room.

7. Carbone, R. & Bean, C., 2011. Generating computer forensic super-timelines under Linux.

8. Buchholz, F. & Falk, C., 2005. Design and Implementation of Zeitline: a Forensic Timeline Editor. DFRWS.

9. Netherlands Forensic Institute (NFI Labs), 2010. Aftertime. http://www.holmes.nl/NFIlabs/Aftertime/index.html.

10. Lowman, S., 2010. Web History Visualisation for Forensic Investigations. MSc Thesis, Strathclyde University.

11. Khan, M., Chatwin, C. & Young, R., 2007. A framework for post-event timeline reconstruction using neural networks. Digital Investigation, 4, pp.146–157.

12. Zhu, Y., James, J. & Gladyshev, P., 2009. A comparative methodology for the reconstruction of digital events using Windows restore points. Digital Investigation, 6(1-2), pp.8–15.

13. Gladyshev, P. & Patel, A., 2005. Formalising Event Time Bounding in Digital Investigations, IJDE, 4(2), pp.1–14.