Forensic Zero-Knowledge Event Reconstruction on Filesystem Metadata

Sven Kälber, Andreas Dewald, Steffen Idler
Department of Computer Science
Friedrich-Alexander-University Erlangen (FAU)
Martensstr. 3
91058 Erlangen, Germany

Abstract: Criminal investigations today can hardly be imagined without the forensic analysis of digital devices, regardless of whether it is a desktop computer, a mobile phone, or a navigation system. This not only holds true for cases of cybercrime, but also for traditional delicts such as murder or blackmail, and also private corporate investigations rely on digital forensics. This leads to an increasing number of cases with an ever-growing amount of data, that exceeds the capacity of the forensic experts. To support investigators to work more efficiently, we introduce a novel approach to automatically reconstruct events that previously occurred on the examined system and to provide a quick overview to the investigator as a starting point for further investigation. In contrast to the few existing approaches, our solution does not rely on any previously profiled system behavior or knowledge about specific applications, log files, or file formats. We further present a prototype implementation of our so-called zero knowledge event reconstruction approach, that solely tries to make sense of characteristic structures in file system metadata such as file- and folder-names and timestamps.

1 Introduction

The data deluge, investigators are facing in digital forensic investigations nowadays, is remarkable and often leads to a time-consuming analysis. Selective imaging [Tur06, SDF13] is considered a possibility of reducing the amount of data to be processed, while abstraction, as well as automation, are regarded to be necessary for investigation speedup [Gar10]. Well known investigative process models [Cas11, Car05] yield starting points for automation during acquisition and extraction of digital evidence and during event reconstruction. Today, commonly used tools already provide automation for the acquisition and extraction of digital evidence, but lack features for automatic event reconstruction.

1.1 Motivation

Up to now, there are only few approaches to automated event reconstruction. One approach is to identify characteristic patterns (digital fingerprints) that emerge in filesystem timestamp metadata while running a specific application or action [KDF13, JGZ11]. Another approach is having individual parsers or analyzers that operate on specific application logfiles and data stores such as sqlite databases [HP12]. While the approach of fingerprinting is more generic, since it is based only on filesystem metadata, the approach of logfile parsing is more specific because each application implements its own logfile format. Thus, investigators are in need of an individual parser for every application of interest. More importantly, the logfile format might change over time, therefore multiple parsers may be needed for different appliction versions. The concept of digital fingerprinting on the other side, similar to classical forensic fingerprint comparison, requires a previously aquired set of known fingerprints. For this reason, prior to actually matching fingerprints in a digital investiagtion, a costly and time consuming profiling phase for each application is required.

1.2 Contributions and Outline

In this paper, we examine the feasibility of automatic event reconstruction that spares prior knowledge of application specifics. We call this approach *zero-knowledge event reconstruction*, since it does not depend on any previous profiling or logfile parsers.

Our approach is to initially extract all filesystem timestamps of a given storage device, then cluster accumulations of timestamp values around different points in time to events. Based on storage location, file names and types, as well as the type of timestamp, these events are then labeled on a best guess basis. To support the investigator during the investigation and to provide a quick overview of all events found this way, those events are being visualized in a dynamically browsable graphic timeline. Moreover, the timeline contains details about which timestamp pattern led to the identification of each specific event.

In particular, we make the following contributions:

- We introduce a novel approach to automated event reconstruction that does not require training phases or application specific analyzers. (Section 3)
- We give a prototype implementation of our approach called *0KER* that identifies events based on timestamp information stored in the filesystem. (Section 4)
- We evaluate the chances and limitations of our approach using the prototype implementation and a compare it to our former approach. (Section 5)

1.3 Related Work

James et al. [JGZ11] generate timestamp signatures for identifying application startups in post-mortem digital investigations. Those signatures are created by first monitoring file access and the Windows Registry with the Microsoft Process Monitor during startup. In a second step, the timestamp update behavior for all files identified with the Process Monitor, is then monitored and saved a signature for the corresponding application startup.

In previous work [KDF13], we have developed a forensic fingerprinting framework named

Py3xF that allows investigators to generate fingerprints for automated event reconstruction. We focused not only on the startup process of different applications but also on application specific actions like sending an email / instant message or bookmarking a website. *Py3xF* allows to automatically create application profiles based on timestamp modifications that occur during application runtime. Based on these application profiles, fingerprints are generated that consist of only unique timestamp modifications. Those fingerprints can than be matched against extracted metadata of a filesystem. All matching patterns are then reported back to the investigator. This way, only known and already profiled / fingerprinted applications and actions may be identified and reported.

Gudjonsson [Gud13] developed the well-known super-timelining tool *log2timeline*. It parses known log files such as windows event logs or apache web server logs. Results can be visualized in timelines using different timeline interfaces like Computer Forenics TimeLab, BeeDocs or SIMILE. However, this approach is very application specific, since individual log file parsers need to be implemented. Moreover, investigators might still be confronted with an overwhelming amount of information, since *log2timeline* does not automatically cluster multiple low-level events to a smaller amount of more meaningful high-level events.

Hargreaves and Patterson [HP12] present an approach for reducing the amount of data by identification of high-level events in automated timeline reconstruction. *PyDFT*, their implementation of a digital forensic timeline generator operates in two stages. In the low-level event extraction stage, filesystem timestamps and timestamps from log files and registry hives are extracted. In the high-level event reconstruction stage, specific analyzers are used to re-create more meaningful high-level events such as Google searches and USB device connection. This approach reduces the data deluge that investigators are facing with tools like *log2timeline* but additionally to extractors, handcrafted sophisticated analyzers are required.

After we have now discussed related work in the field of automated event reconstruction and noticed that all approaches depend on prior knowledge of application specifics in either handmade parsers / analyzers or automatically profiled / fingerprinted timestamp patterns, we detail our approach of *zero-knowledge event reconstruction* including some required background information in the following sections.

2 Background

The basic idea of this work is to extract the timestamp information stored in the filesystem of a seized hard disk and then try to automatically reconstruct certain events (such as application startups) that recently happened on the system it was taken from. This is done by clustering all files that were accessed, modified or created around certain points in time. Since we set our focus on the NTFS filesystem, we will now briefly describe the specifics of NTFS timestamps before we introduce the applied string metric and clustering algorithms that are used for event reconstruction.

2.1 Timestamps in the NTFS filesystem

Microsoft's New Technology File System (NTFS) is used as the default filesystem in Windows NT, 2000, XP, Vista, 7 and 8, making it the most widely used filesystem on private computers nowadays. Besides other metadata, like the filename and size, NTFS keeps track of the four following timestamps:

- atime: the time, a file / folder was last accessed.
- mtime: the time, a file / folder was last modified.
- crtime: the time, a file / folder has been created.
- ctime: the time, the metadata entry of a file itself was last updated.

2.2 Levenshtein Distance

The Levenshtein distance is a string metric introduced by Vladimir Levenshtein [Lev66]. For two given strings the Levenshtein distance is indicating the minimum amount of edit operations required to transform the first string into the second. Thereby, the possible edit operations are insertion, deletion or substitution of a single character in the string. As a simple example, we want to calculate the Levenshtein distance between the the two strings `a = 'house'` and `b = 'spouse'`. Since the string length of string `a` is 4 and of string `b` is 5, we obviously need to insert one character in string `a`. Thus, the Levenshtein distance is at least 1. So, if we insert an `s` at the beginning of string `a` and then substitute the `h` with an `p`, we successfully transformed string `a` into string `b`. For this transformation, we required at least an insertion and a substitution operation and therefore the Levenshtein distance between `'house'` and `'spouse'` is 2.

2.3 Clustering with DBSCAN

Ester et al. [EKSX96] created the well-known and broadly used data clustering algorithm *DBSCAN*. Figure 1 depicts the concepts of *density-reachability* and *density-connectedness* in DBSCAN. DBSCAN requires the following two parameters:

- `eps`: the maximum distance between two objects
- `minPts`: the minimum number of objects required to create a cluster

Two objects are considered to be directly density-reachable if the distance between both objects in not greater than the distance defined by `eps` and there is a sufficient number (`minPts`) of other objects in the cluster. Object `m` is thus directly density-reachable from object `p` for `minPts = 4` in our example. Object `q` is not directly density-reachable from object `p` but is from object `m`, but still `q` is considered to be density-reachable from `p`, since there is a link of directly density-reachable objects from `p` to `q`. Objects like `s` or `t` on the edge of a cluster are considered to be density-connected since they both are density-reachable from object `q`, even if there is not a sufficient number (`minPts`) of objects within their maximum distance (`eps`). A cluster of objects in DBSCAN consists of those objects that are all density-connected.

After we discussed the basics of our approach, the following section covers the details of *zero-knowledge event reconstruction*.

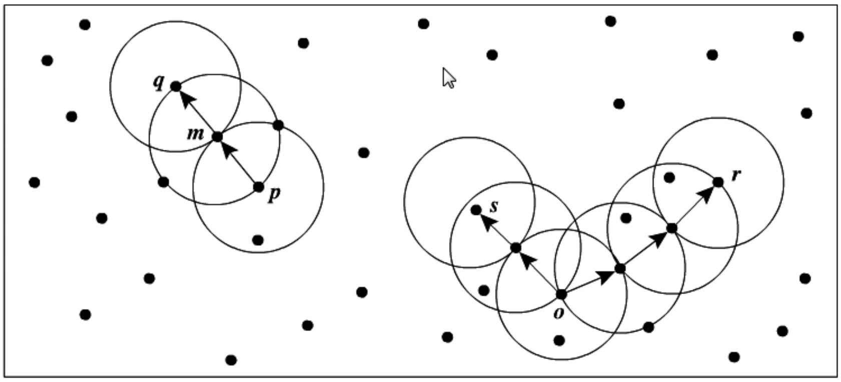


Figure 1: *density-reachability* and *density-connectedness* in DBSCAN [Han12]

3 Zero-Knowledge Event Reconstruction

As discussed previously, automatic event reconstruction approaches either require an up front, time consuming profiling phase or an individually written parser or analyzer so far. Either way, prior knowledge of application specifics such as file storage location, log file format or timestamp modification patterns are required.

Our approach is to spare prior knowledge on the application level, by utilizing the filesystem's timestamp information update behavior on a significant amount of files for each user triggered activity. Grier [Gri11] observed that under normal usage filesystem activity is neither uniformly nor normally distributed over files but is following a Pareto distribution. Thus, we argue that for individual actions, their execution lead to timestamp metadata updates on a significant amount of files stored in the filesystem. For example, when starting an application, not only the corresponding binary file is accessed but many other files such as linked libraries, configuration files, images, media files or log files are accessed or even modified or created. In case of such an event, the filesystem then updates the timestamp information accordingly for each file. Since those file accesses happen close to each other in terms of time, their corresponding timestamp values do not differ by more than a few seconds from one another.

The idea is to extract all timestamp information and the corresponding file name including the absolute path from a filesystem of interest and sort them by their timestamp values in ascending order. This basically results in a timeline of all file creations, file modifications and file accesses of the entire volume. Since different applications and events mostly access different files and because of the Pareto distribution, peaks (in terms of amount of files) arise on the timeline. In other words, each peak corresponds at least to one event that caused these different timestamp updates. Therefore and for simplicity, we refer to such peaks as events in the next sections.

Each event consists of different files that were accessed, modified or created during the same time. Therefore, files that belong to the same application and are located in the same locations (e.g. in the same subdirectory) are clustered. Since, application specific files may additionally be distributed over different locations in the filesystem, multiple clusters might get created for each event. For instance, application data is most of the time stored apart from operating system libraries, temporary files, or application configuration data.

Moreover, the type of timestamp update is being analyzed further, because this can give an explanation about the actual event and the modification of files. Based on these clusters and timestamp analyses, a label is created for each event, indicating which application was used or which files were accessed or modified. These resulting events can then be visualized in a timeline fashioned style, to support the investigator in the event reconstruction phase, by providing a quick and easy overview of all identified events.

4 Prototype Implementation

To demonstrate our approach of *zero-knowledge event reconstruction* we implemented a prototype called `0KER.py` (0-Knowledge Event Reconstruction) in Python. In the following sections, we have a closer look at the details of the three main stages of `0KER.py`: the detection of timestamp structures, event labeling and the timeline visualization.

4.1 Detection of Events in Timestamp Structures

`0KER.py` operates on XML representations of filesystem metadata generated by *fiwalk* [Gar09]. In a first step, the *fiwalk* XML report is parsed and all file names including their path and timestamp information is extracted. Then, in a second step, we sort this data by their timestamp values in ascending order, before the events then are determined by aggregating those files that have been accessed, modified or created sequently. Whenever a gap between two consecutive timestamp values of more than 20 seconds is detected, a new event is being created. Our evaluation has shown, that 20 seconds was, at least in our case, the value leading to optimal results. On 20 different disk images we observed, that for values smaller than 20 seconds the amount of events that were mistakenly divided into two or more events rises. While, for values greater than 20 seconds, the amount of events that mistakenly contain traces of two or more events increases. Moreover, this approach has shown to be more suitable for event detection than clustering algorithms like k-means.

In a pre-evaluation, we have observed, that clustering algorithms that are based on partitioning are not well suited for automatic event reconstruction, since k-means, k-medoids or other alike algorithms aim to cluster `n` objects in `k` clusters. Even though `k` is variable among multiple runs, a fixed value needs to be chosen before each clustering run. Since the amount of events is unknown to the investigator prior to the investigation, clustering based on partitioning is not applicable. On the other hand, due to the sparseness of timestamp values and the Pareto distribution observed by Grier [Gri11], the approach of identifying events based on the distance of timestamp values among each other, has shown to be well suited.

4.2 Event Labeling

The labeling of events is done by analyzing the file types in combination with the timestamp type of all files within each event. If a pattern of timestamp types, according to the rules depicted in Figure 3 is found, the corresponding event description is assigned to the appropriate file. This way, newly created files, copied files as well as files that have been modified during a specific event are identified and reported to the investigator. Additionally, known file types like executables, documents, media files, audio files and databases are analyzed separately. For example, if a file access on an executable file is detected, the event is labeled as “Application `app.exe` started”. If two or more events on a system happened around the same time, the chance is that those events are regarded as a single event due to the event detection mechanism described above. In such cases, an event contains for example multiple application startups. Therefore, we still list all generated labels as a result, so that the investigator is able to notice both events.

Moreover, the timestamp changes within each event are clustered according to their file names and paths. This is done, in order to group all files in the same storage locations. The following section describes the clustering in more detail.

4.2.1 Path & Filename Clustering

The clustering is done in two steps. In the first step, we apply the concept of string metrics by calculating the Levenshtein distances for the string representations of the absolute path (including the filenames) of all files that have been accessed around the same point in time. Thereby, smaller values indicate files that are located in the same subdirectories or share similar folder structures. Thus,we are able to create a distance matrix which is, in the second step, used as input for the DBSCAN algorithm. As a result of DBSCAN, all files that have a small Levenshtein distance and thus share the same subdirectories for example, are clustered. These clusters ease the analyzing process for investigators and reduce the amount of data to be sighted, by consolidating multiple timestamp updates that occurred during the same time and belong to the same application.

For example, the start of an application causes program data located in `\Program Files \application\` to be accessed. Additionally, shared libraries are loaded from `\Windows \system32\` and configuration files stored in `\Users\username\AppData\` are read. In this example, the following three clusters would be identified by DBSCAN for this event. One cluster containing all files accessed in the applications installation folder. The other cluster contains the shared libraries of the Windows installation folder and a third cluster is

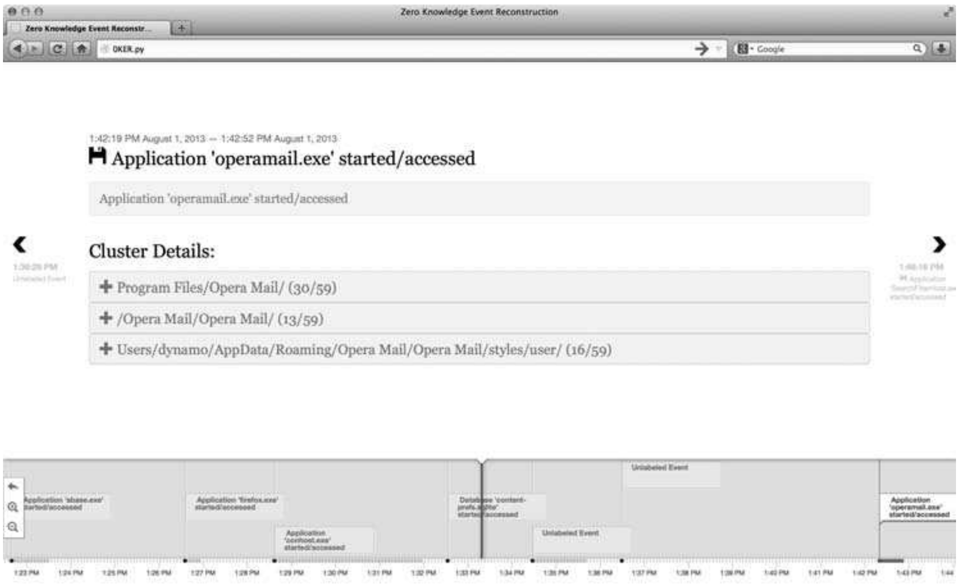

Zero Knowledge Event Reconstruction
Zero Knowledge Event Reconstr...
0KER.py
Google
1:42:19 PM August 1, 2013 — 1:42:52 PM August 1, 2013
Application 'operamail.exe' started/accessed
Application 'operamail.exe' started/accessed
Cluster Details:
Program Files/Opera Mail/ (30/59)
/Opera Mail/Opera Mail/ (13/59)
Users/dynamo/AppData/Roaming/Opera Mail/Opera Mail/styles/user/ (16/59)
Application 'sbase.exe' started/accessed
Application 'firefox.exe' started/accessed
Application 'conhost.exe' started/accessed
Database 'content-prefs.sqlite' started/accessed
Unlabeled Event
Unlabeled Event
Application 'operamail.exe' started/accessed
1:23 PM
1:24 PM
1:25 PM
1:26 PM
1:27 PM
1:28 PM
1:29 PM
1:30 PM
1:31 PM
1:32 PM
1:33 PM
1:34 PM
1:35 PM
1:36 PM
1:37 PM
1:38 PM
1:39 PM
1:40 PM
1:41 PM
1:42 PM
1:43 PM
1:44

Figure 2: Visualization of events found with 0KER.py

created for the user specific application data stored in the user's home folder. Additionally, for each cluster the amount of files that have been accessed, created or modified is counted. Using this information, the investigator is able to quickly identify the important clusters for each event. Moreover, two events that happened simultaneously and thus were regarded as a single event, are easier to distinguish by investigators since files from one application are grouped in other clusters than files from another application.

As described in Section 2.1, DBSCAN requires two parameters, namely *minPts* and *eps*. Where *minPts* determines the minimal amount of objects required to form a cluster and *eps* indicates the maximum distance between two objects in the same cluster. More precisely, in our case *eps* is set to the maximal distance, measured as Levenshtein distance, the absolute path (including the filename) of two files may have in order to be contained in the same cluster. *minPts*, on the other hand, indicates the amount of timestamp updates of different files that are required in order to create a new cluster. In total, we evaluated 12 different combination sets of *eps* and *minPts* on 6 different images. In our case, the optimal value for *minPts* is 12 and 50 for *eps*. These values led to the best results in terms of detection rates, amount of events and correctly labeled events.

Additionally to the clustering of files, we further analyze the type of timestamp modification. We highlight files that were created, updated, copied from another location or renamed by checking if the timestamp values found on disk match a known pattern as depicted in Figure 3. This is done to provide the investigator a quick overview of how many files were accessed, which files were newly created or which files have been modified during this event and where they are located. The amount of files accessed might help the investigator to verify the automatic identification of this event as well as to distinguish two independent events that were mistakenly identified as one single event.

4.3 Visualization

Timelines are intended to support the investigator in early stages of the event reconstruction phase of a digital investigation. Our goal is to provide a quick and easy overview of all found and identified events that may point the investigator to suspicious files, applications, system behavior or other activities for further in-depth analysis.

Figure 2 depicts a timeline created with `0KER.py` and visualized by TimelineJS [Ger13]. TimelineJS is an open source, javascript, json and html based timeline visualization tool created by the Northwestern University Knight Lab. Each event is marked in a balloon-shaped box in the timeline view located at the bottom of the page. The yellow-colored info box holds the identified label for the activity. In case multiple labels exist for an event, all labels are listed, to indicate the investigator that possibly two or more activities happened at that point in time. For each identified cluster of files a grey-colored, collapsible box is created, that is labeled with the longest common substring of all path names within the cluster. Upon expanding a cluster, a list of all files and subfolders that were accessed, modified, or created around the same time is shown. With this detailed information, the investigator is able to understand and verify the results of the automatic event reconstruction done by `0KER.py`.

In the following section, we are now going to evaluate our implementation of `0KER.py` and compare our approach to the results of the fingerprinting framework *Py3xF*.

5 Evaluation

For our evaluation, we installed Microsoft Windows 7 along with 10 different applications on two independent virtual machine disk images. Oracle's VirtualBox was used as virtualization platform and the installed applications are: *Mozilla Firefox, Mozilla Thunderbird, Opera Browser, Opera Mail, Google Chrome, Pegasus Mail, OpenOffice, LibreOffice, ICQ* and *WinSCP*. On each machine, we then automatically and consecutively started 5 different applications and denoted the time of execution. After powering off the machines, we extracted the file system's timestamp metadata from both images using *fiwalk*. Next, we started `0KER.py`. As a result, for the first disk image, 7 events and for the second image 8 events were detected and reported. Table 1 and Table 2 show which actions were found with `0KER.py` in comparison with *Py3xF*. As can be seen in Table 1, `0KER.py` correctly identified, matched and labeled the startup process of *LibreOffice Base, Mozilla Firefox* and *Opera Mail*, which is indicated by the checkmark (✓). The startups of *ICQ* and *Google Chrome* on the other hand, were marked with an **o**, since those events could not be labeled correctly. This is the case, whenever the accessed timestamp of the according application executable is updated in later points of time for example. However, these events were not marked as unidentifiable (✗), since the report contains at least one event entry for both startup processes with significant clusters of timestamp updates for the corresponding application, indicating that the application was in fact running during that point in time. *Py3xF* was able to correctly identify and label all but the Mozilla Firefox startup process, based on fingerprints that have previously been generated for these 10 applications. In this case, *Py3xF* reported an action performed with Mozilla Thunderbird instead.

On the second image, we not only performed the startup process for the individual applications, but we also send out an email using Mozilla Thunderbird. As can be seen in Table 2, *Py3xF* was able to correctly identify and label all performed application startups and the event of sending an email with Thunderbird. `0KER.py` on the other hand, failed to identify the start of WinSCP. The event of sending an email was reported in three different events and may be reconstructed by the investigator. Two events were created stating, that the database files `addons.sqlite`, `extensions.sqlite` and `places.sqlite` of Thunderbird were modified. But more importantly, an event was created that indicated the access and modification of the IMAP store of the corresponding Thunderbird email profile in `AppData/Roaming/Thunderbird/Profiles/fsk381mz.default/ImapMail/imap.googlemail.com/[Gmail].sbd/`.

As mentioned earlier, although we only performed 5 actions on each disk , 7 events were reported for the first image and 8 for the second one. This is because on the first image the events for starting Mozilla Firefox and Google Chrome were split up in two events each. On the second image the additional events were created for the event of sending an email in Mozilla Thunderbird.

As we can see, with our zero-knowledge event reconstruction approach, it is possible to automatically identify and label most application startup events correctly.

Event	0KER.py	Py3xF
LibreOffice Base	✓	✓
Mozilla Firefox	✓	✗
ICQ	o	✓
Google Chrome	o	✓
Opera Mail	✓	✓

Table 1: Results of disk image 1

Event	0KER.py	Py3xF
Thunderbird/send mail	✓/o	✓/✓
Pegasus Mail	✓	✓
OpenOffice Calc	✓	✓
WinSCP	✗	✓
Opera Browser	o	✓

Table 2: Results of disk image 2

However, application specific activities such as sending an email can only be matched by tools based on prior knowledge of the application or have to be identified by the investigator. With the concept of clustering timestamp updates around certain points in time, based on their file name and path, we introduced a possibility to reduce the amount of data to be analyzed by investigators in order to identify such application specific activities.

6 Conclusion

In this paper, we have presented an approach to event reconstruction in post-mortem investigations that does not depend on application specific parsers or fingerprints. A prototype implementation called `0KER.py` was developed for Microsoft Windows systems based on the NTFS filesystem.

With our implementation, which identifies timestamp update aggregations around certain points in time, we were able to reconstruct particular user activities and highlight events that occurred but could not be tied to a specific application. With this approach, it is possible to reconstruct events such as application installations and startups, as well as access and modification of application specific files like configuration files, temporary files or data stores. Whereas the identification of application specific user activities such as sending or receiving an email, that may be automatically identified by approaches like *Py3xF*, still requires expert knowledge of investigators.

6.1 Limitations

Obviously, the timeline generated by `0KER.py` only contains the latest occurrences of different events, since NTFS only stores the latest timestamp values in the $MFT and there is no history of previous values. Additionally, in cases, when two or more events happen simultaneously or right after another, those might be mistakenly considered and shown as one single event in the timeline. This is due to the nature of the basic approach of event detection discussed in Section 4.1. Moreover, compared to application specific approaches like *log2timeline* or *Py3xF*, that are able to also identify individual user activities with applications, `0KER.py` might only label events for application startup or installation as well as document and database access and modifications. Nevertheless, by clustering, `0KER.py` greatly reduces the amount of events that has to be sighted by an investigator and highlights timestamp update aggressions on the timeline.

References

[Car05] Brian Carrier. *File System Forensic Analysis*. Addison-Wesley Professional, 2005.

[Cas11] Eoghan Casey. *Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet*. Academic Press, third edition, 2011.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[Gar09] S. Garfinkel. Automating disk forensic processing with SleuthKit, XML and Python. In *Proceedings of the 2009 Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering*, pages 73–84, 2009.

[Gar10] Simson Garfinkel. Digital Forensics Research: The Next 10 Years. *Digital Investigation*, 7:64–73, 2010.

[Ger13] Joe Germuska. Timeline JS - Beautifully crafted timelines that are easy, and intuitive to use. `http://timeline.knightlab.com/`, November 2013.

[Gri11] Jonathan Grier. Detecting data theft using stochastic forensics. *Digital Investigation*, 8:S71–S77, August 2011.

[Gud13] Kristinn Gudjonsson. log2timeline `https://code.google.com/p/log2timeline/`, November 2013.

[Han12] Jiawei Han. *Data mining : concepts and techniques*. Morgan Kaufmann series in data management systems. Morgan Kaufmann/Elsevier, Waltham, MA, 3rd ed edition, 2012.

[HP12] Christopher Hargreaves and Jonathan Patterson. An automated timeline reconstruction approach for digital forensic investigations. *Digital Investigation*, 9, Supplement(0):S69 – S79, 2012. The Proceedings of the Twelfth Annual {DFRWS} Conference.

[JGZ11] Joshua Isaac James, Pavel Gladyshev, and Yuandong Zhu. Signature Based Detection of User Events for Post-mortem Forensic Analysis. In *Digital Forensics and Cyber Crime*, volume 53 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 96–109. Springer Berlin Heidelberg, 2011.

[KDF13] Sven Kälber, Andreas Dewald, and Felix C. Freiling. Forensic Application-Fingerprinting Based on File System Metadata. In *IT Security Incident Management and IT Forensics (IMF), 2013 Seventh International Conference on*, pages 98–112, 2013.

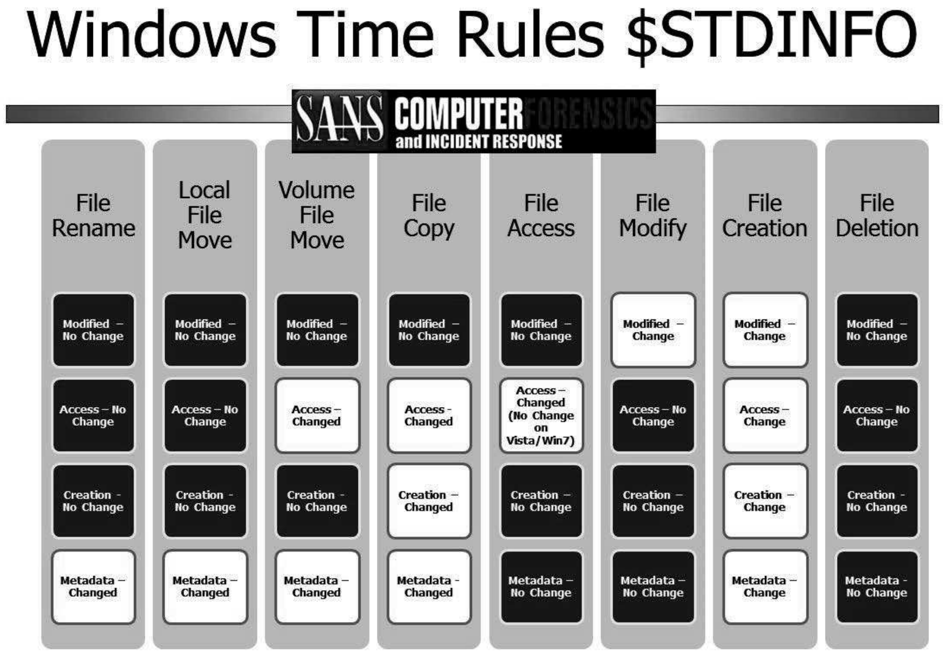
[Lee10] Rob Lee. Windows 7 MFT Entry Timestamp Properties `http://computer-forensics.sans.org/blog/2010/04/12/windows-7-mft-entry-timestamp-properties`, April 2010.

[Lev66] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8, 1966.

[SDF13] Johannes Stüttgen, Andreas Dewald, and Felix C. Freiling. Selective Imaging Revisited. In *7th International Conference on IT Security Incident Management & IT Forensics (IMF)*. IEEE, 2013.

[Tur06] P. Turner. Selective and intelligent imaging using digital evidence bags. *Digital Investigation*, 3:59–64, 2006.

Appendix



Windows Time Rules $STDINFO

SANS COMPUTER FORENSICS and INCIDENT RESPONSE

File Rename	Local File Move	Volume File Move	File Copy	File Access	File Modify	File Creation	File Deletion
Modified – No Change	Modified – No Change	Modified – No Change	Modified – No Change	Modified – No Change	Modified – Change	Modified – Change	Modified – No Change
Access – No Change	Access – No Change	Access – Changed	Access - Changed	Access – Changed (No Change on Vista/Win7)	Access – No Change	Access – Change	Access – No Change
Creation - No Change	Creation - No Change	Creation - No Change	Creation – Changed	Creation – No Change	Creation – No Change	Creation – Change	Creation - No Change
Metadata – Changed	Metadata – Changed	Metadata – Changed	Metadata - Changed	Metadata – No Change	Metadata – No Change	Metadata – Change	Metadata - No Change

Figure 3: Windows Time Rules [Lee10]