Automated Collection and Correlation of File Provenance Information

Ryan Good, Gilbert Peterson

Chapter 15

AUTOMATED COLLECTION AND CORRELATION OF FILE PROVENANCE INFORMATION

Ryan Good and Gilbert Peterson

Abstract The provenance of a file is a detailing of its origins and activities. Tools have been developed that help maintain the provenance of files. However, these tools require prior installation on a computer of interest before and while provenance-generating events occur. The automated tool described in this chapter can reconstruct the provenance of a file from a variety of artifacts. It identifies relevant temporal and user correlations between the artifacts and presents them to an investigator. Results from six use cases demonstrate that these correlations are reliable and valuable in digital forensic investigations.

Keywords: File provenance, Windows operating systems, forensic timelines

1. Introduction

Computer forensics, which involves analyzing a digital medium for evidence of a crime, requires the tracking and digesting myriad files and their relationships. Parsing this information can be a daunting task and the time requirement to conduct an analysis can prevent an investigator from obtaining the information needed to prosecute a crime. The automated extraction of the relationships between files and their origins, along with the number of times and ways in which they have been modified and accessed, can greatly speed up this process.

The provenance of a data object (e.g., file) is the "ownership and the actions performed on [the] data object" [7]. Ownership describes the creator of the file or the user responsible for the file arriving on the system, while the actions describe how the file was interacted with post arrival. In many cases, it is important to discover the responsible party for the arrival of a file on a system in order to determine attribution.

Incorrectly arriving at the heredity of a file or not providing enough evidence can derail a case.

The automated tool presented in this chapter enables a digital forensic investigator to quickly identify correlations that can determine the source and activity of a file in a system image. This is accomplished by extracting common sources of provenance information from a storage media image. The automated tool then processes and compares the extracted information to determine the correlations that exist. The correlations are provided to the digital forensic investigator to assist with the case.

Six use cases demonstrate the efficacy of provenance information extraction and correlation. The use cases cover a range of file sources as well as common file activities. The correlations that are discovered are listed within certain categories. A short explanation is provided for each use case along with the results, demonstrating how the automated tool can help a digital forensic investigator identify the provenance of files.

2. Related Work

This section discusses methods for gathering file provenance information. It also provides an overview of the locations that may contain temporal and ownership artifacts in a Windows operating system. Finally, it discusses research in the area of automated provenance creation.

Provenance refers to the earliest known history of an object [7]. It can also refer to the record of ownership of an object. In the context of a computer system or network, provenance pertains to the origins of a piece of data, its relationship to other pieces of data and the processes that created and modified it. One use of provenance is as metadata, which enables a user to search for a file based on past interactions or the original source. A user may forget the document that he/she was working on, but may remember that email was exchanged with someone about pertinent data. This information can help reduce the search space of possible documents, enabling the user to quickly identify the object of interest.

2.1 File Provenance Maintenance Systems

A file provenance maintenance system tracks and gathers file provenance information. The system runs in the background and monitors and records all file actions performed on the system. For example, the Provenance Aware Storage System (PASS) [10] collects and maintains provenance information comprising references between files and memory

Table 1. Temporal Granularity.

Target	Source	Granularity
Registry Last Modified Times	`NTUSER.dat`	Microseconds
File MAC Times	File of Interest	Seconds
History Entries	Browser History Files	Seconds
Recent Documents	`NTUSER.dat`	Days
USB Key	SYSTEM	Seconds
User/Group Information	SAM	Seconds
CurrentVersion Subkey	SOFTWARE	Seconds

elements such as pipes and sockets in order to create the provenance of files.

The File Provenance System (FiPS) [14] enables the recreation of files. This system improves on the significant overhead required by PASS. However, it still suffers from overhead due to its use of a stackable filesystem. Stackable filesystems are much easier to develop than kernel-level filesystems. Unfortunately, this ease of use comes at the cost of performance [15].

2.2 Sources of Provenance Data

Sources of file provenance information in Windows operating systems running the NT Filesystem (NTFS) include the modified, accessed and created (MAC) times, file metadata, Windows registry hives and application history, and log files. Table 1 lists the sources of provenance data and their temporal granularities.

NTFS MAC Times and File Metadata. NTFS stores the modified, accessed and created (MAC) times for each file in the Master File Table (MFT) [8]. The `mtime` is the time of a file's last modification; it updates whenever the file contents change. The `ctime` updates whenever a file's content changes; it also updates whenever the file attributes change. A file's attributes can change for many reasons, including file movement and ownership changes. The `atime` is the last interaction time of a file and updates after any type of interaction, including simply opening the file.

To summarize, if a file is simply opened and viewed, only its `atime` changes. If the file is opened, viewed and edited, the `atime` and `mtime` change. If the file is opened, edited and placed in another directory, then all three values change. It is also important to note that all three values change when the file is copied and pasted. This is because copying and

pasting creates a new file. This does not occur if the file is simply moved because it is still the same file. When a file is copied or moved, the MAC times of the containing folders reflect similar changes.

Many file formats contain metadata, which is data about data. File metadata may comprise the file creator, the subject that last modified the file and when actions on the file occurred. It may be tempting to simply accept these values and assume that the origins of the file are known. Unfortunately, values may be missing and are easily modified. Therefore, a digital forensic investigator should either fill in the missing information or validate the available data. Metadata can be acquired in a number of ways; the ExifTool [6] was used to extract metadata in this work.

Registry Hives. The registry is also a source of provenance for a filesystem. The registry contains a number of hive files that have responsibilities ranging from tracking user activities to holding system configuration information. Whenever an event occurs in a filesystem, it can be expected to impact the registry in some way. Every registry key has a value known as the "last write time," which is modified when a relevant event occurs. This value is extracted by timeline generators such as `log2timeline` [4] when they collect temporal artifacts from storage media. RegRipper [13] is also a useful tool for searching the registry and collecting items of interest.

The registry has two types of hives: (i) user hives; and (ii) system hives [2]. User hives focus on specific users and contain data that can be used to trace user activity. User hives include `NTUSER.dat` and `USRCLASS.dat`. System hives include the Security, SAM, System, Software and AmCache hives. System hives contain information about the overall functioning of the computer system. Information about external storage connections, user login dates and times, account permissions, program executions, etc. are easily obtained from the hives using tools such as RegRipper.

Application History Files. Web browser applications serve as entry points to a file. Browser application history files exist for each user on a system as well as for each browser employed by a user. Each file contains the recent history of a user's activities involving a specific browser. Internet Explorer, Google Chrome and Mozilla Firefox have separate history files that are viewable with the right tools. The history files can be parsed to discern activities that occurred in close temporal proximity to the file's arrival on the system. This can help determine if the file arrived on the system via download from a website.

Each browser requires a separate tool to parse its history. NirSoft provides a suite of tools that includes Internet Explorer History View (IEHV) [11] for viewing Internet Explorer history folders, ChromeHistoryView [12] for the Google Chrome browser and MZHistory View, a Firefox history viewer.

2.3 Evidence Correlation

Image analysis tools such as EnCase, Sleuth Kit and Forensic ToolKit can be used to gather event information, but they do not present it in a form that facilitates the comparison of timestamps. Timelines are a visualization aid that assist forensic investigators in understanding the events that occurred on a system. Zeitline [1] allows for enhanced visualization of the data in a storage media image in the form of timelines. An investigator may import events that are grouped, filtered and presented in a manner that is more indicative of a timeline or sequence. The `log2timeline` tool expands on this functionality by incorporating additional sources of timeline data such as artifacts and log files. Timeline tools facilitate information presentation, but still require effort on the part of a human to parse the events and determine correlations. PyDFT [5] improves on the basic timeline functionality by analyzing low-level events to determine when high-level interactions (e.g., USB device connections) occur. The functionality provided by these timeline tools is valuable, but they do not provide summarized information about a file of interest.

A clear and concise view of relevant data makes it easier for digital forensic professionals to quickly parse through information that can aid in their investigations. The FACE tool [3] enhances data presentation and automates forensic data correlation; it primarily focuses on data in system memory and network traffic captures. Ramparser, a tool for Linux memory analysis, gathers information from running processes, open files and socket/netstat information. This data is provided by Ramparser to FACE, which then discovers correlations. Unfortunately, none of this is viable for storage media images.

Forensic automation is also valuable for determining the attributes of a file. The approach leverages machine learning techniques to sift through large amounts of data. For example, a machine learning algorithm can use provenance data to determine a file's extension [9]. This data includes the relationships between files and processes, their locations relative to each other and the frequency with which they are accessed.

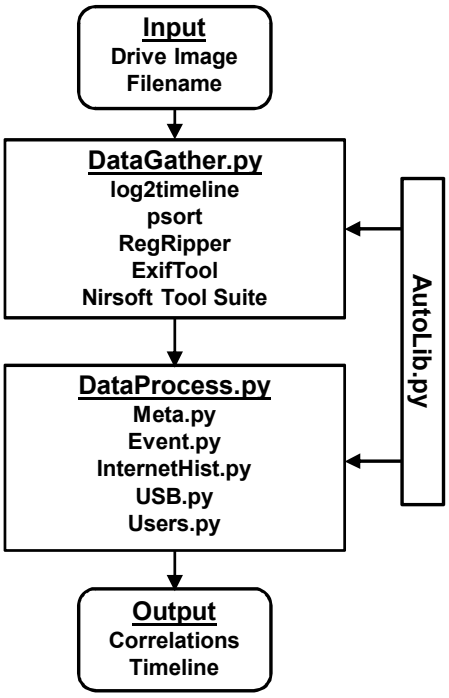

Figure 1. Provenance Collection Tool.

3. Provenance Collection

The Provenance Collection Tool developed as part of this research constructs the provenance of a file of interest and its logical path in a storage media image. The tool runs several external programs to collect temporal and association artifacts related to the file. It then parses the information to discover correlations that can help determine the origin of the file and activities involving the file.

Figure 1 shows the provenance extraction and correlation process. Data is collected by `DataGather.py`, which invokes the RegRipper, `log2timeline`, `psort`, ExifTool, ChromeHistoryView, MZHistory View and IEHV tools. Next, `DataProcess.py` attempts to determine the origins of the file in the image. It accomplishes this by searching for indicators that a forensic investigator often leverages to determine file origin. The two Python scripts use functions in `AutoLib.py`, a library created as part of this research. The Provenance Collection Tool considers indicators related to local file creation, web browsers, USB devices, etc. The indicators are passed to the user, who can draw conclusions and determine how to best tailor the forensic analysis to the investigation.

3.1 Data Gathering

`DataGather.py` requires the file of interest to be specified by its name and logical path, along with the location of the image in which it is contained. In order to mount the image, `DataGather.py` needs to know the start block of the image of interest. It obtains this information by running `mmls` on the image and routing the output to a text file, which is parsed to find the start block of the image. The block size is assumed to be 512 because this is almost always the case. After the information has been gathered, the `mount` command is invoked to locally mount the image as read only. The `find` command is then invoked to find the file of interest. Following this, `DataGather.py` checks for a file with the same name that has a `.torrent` extension, as this can be an indicator of a torrent source.

File metadata is extracted using ExifTool. This tool collects information about the file creator, editor and creation/editing dates/times, if they are available. The `state` command is then invoked to obtain the NTFS MAC times of the file.

After capturing the metadata, `DataGather.py` obtains the last write times from the `NTUSER.dat` hive, along with any pertinent values. The `NTUSER.dat` hive is examined because it contains most of the information relevant to file provenance, including user activities and program execution. This is accomplished using the `log2timeline` tool, along with filters that prevent it from analyzing the entire image.

For user attribution, `DataGather.py` checks the Users folder in Windows to determine the users that are present in the system. The default users that are present in all systems are filtered along with unrelated directories. These include All Users, `desktop.ini`, Default, Default User and Public.

`DataGather.py` obtains the USB connection history from the system hive. This information includes the first date/time that the system interfaced with the USB device, most recent time that the system interfaced with the USB device and serial number of the device. Following this, the RegRipper `samparse` plugin collects all the relevant information in the SAM hive. Since `log2timeline` gathers information in the GMT/UCT format while other sources are relative to the system time, the system timezone is parsed from the SYSTEM registry hive to obtain the data.

The next item of interest is the user web history. The IEHV tool is used to obtain a user's Internet Explorer history by examining the corresponding history folder. Following this, a similar process occurs with ChromeHistoryView. Finally, the MZHistoryView tool obtains the user's Firefox history.

3.2 Data Processing

A digital forensic investigator typically has to manually parse the collected data in order to determine where relevant correlations may exist. `DataProcess.py` automates many of the correlation checks that are relevant to file provenance by parsing the data and modifying it so that all the data is in the same format and comparable. It then searches for correlations and sets the relevant Boolean flags based on its findings.

`DataProcess.py` facilitates the comparisons by organizing all the data sources into class structures. All timestamps are modified to be in UTC 24-hour time, the names of months are replaced by their numerical equivalents, etc., in order to enable value comparisons.

Five categories of provenance correlations are considered. The categories are: (i) local factors; (ii) browser factors; (iii) USB factors; (iv) Skype factors; and (v) torrent factors.

Local Factors. The following Boolean flags are indicators of local file creation or interactions:

- **recentuser:** The file of interest appears in the user's recent document registry key list.
- **localuser:** A reference to the file exists in the user's recent documents or any recent documents iterations in the history (found using `log2timeline`).
- **date_check:** Relevant timeline entries exist in `NTUSER.dat`, which refer to the file of interest on the day of its creation. The creation date is determined from the metadata, if available. If no metadata is available, the flag is set to false.
- **time_check:** Relevant timeline entries exist in `NTUSER.dat`, which refer to the file of interest within 30 minutes of its creation. The creation time is determined from the metadata, if available. If no metadata is available, the flag is set to false.
- **systemuser:** The file creator has the same user name as another system user. The creating user is determined from the metadata, if available. If no metadata is available, the flag is set to false.
- **word_create_day:** Microsoft Word was used on the creation date of the file. The flag is only set to true if the file type is a Microsoft Word document.
- **word_appear_day:** Microsoft Word was used on the first day that the file was seen on the system. This flag is mostly a fail-safe if

the file metadata is not available. It is also useful to reinforce the validity of the metadata and to indicate the possibility of editing, but not creation. The flag is set to true if Microsoft Word ran on the same day that the file first appeared in the log created by `log2timeline` based on `NTUSER.dat`.

- **impossiblelocal:** The operating system was installed on the system after the file was created. Therefore, the file could not have been created on the system. The creation date/time is determined from the metadata. Therefore, if metadata is unavailable, the flag is set to false.

- **movement:** The file was likely moved within the system file structure. This is determined based on the file's MAC times. If `mtime` is much different from `ctime`, then the file was likely moved because there are few other reasons for this difference in the timestamps.

- **editing:** This indicates possible file editing while the file was on the system of interest. This is based on the file's MAC times as well as `NTUSER.dat` timestamps. If `mtime` is not relatively close to the first date/time the file was seen on the system, then the file has likely been edited. Note that this could also occur if the user had copied and pasted the file of interest and then deleted the original file. This is because the system would consider the copied file as a new file and reset the MAC times to the time when the file was copied while the same arrival date/time would be in `NTUSER.dat`.

- **difmod:** The file was modified by someone other than the creator.

- **samedaylogin:** This flag is set to true if the last login date of a system user is the same as the date that the file arrived on the system. All the users with the matching last login date are listed.

Browser Factors. The following Boolean flags are indicators of browser source:

- **relevant_chrome_visits:** This flag is set to true if Chrome websites were visited within ± 2 hours of the file's arrival on the system. If this is the case, all the relevant visits are recorded. For each visit, the user who visited the site, the site that was visited and the date and time of the visit are listed.

- **relevant_ie_visits:** This flag is the same as that for Chrome, except that it is for Internet Explorer.

- **relevant_firefox_visits:** This flag is the same as that for Chrome, except that it is for Firefox.

USB Factors. The following Boolean flags are indicators of USB source:

- **timelinerelevant_removable_disk_usage:** This flag is set to true if a timeline entry references the file of interest and a USB device. This usually occurs when the recently used documents contain references to both items at any point in time.
- **usbdatematch:** A removable disk was used on the same day that the file first arrived on the system. This is determined based on the output of RegRipper's `USBdevices` plugin, which examines the system hive to determine when USB devices were last used. This flag assumes that, if a user decides to transfer a malicious or inappropriate file to a computer using a USB device, the USB device will most likely not be used again. For this reason, the USB device's last write times are compared against the first sighting of the file in the `log2timeline` logs.

Skype Factors. The following Boolean flags are indicators of Skype source:

- **skypedatematch:** Skype was used on the same day that the file first arrived on the system. This is determined by searching for references to `Skype.exe` under the `UserAssist` key in the registry. An entry is created whenever Skype is used. All updates to this key can be seen in the timeline created by running `log2timeline` on the `NTUSER.dat` hive.
- **skype30min:** Skype was used within 30 minutes of the file first being seen on the system. This is the same check as `skypedatemach` except that the granularity of the check is narrower.

Torrent Factors. The following Boolean flags are indicators of torrent source:

- **torrentfile:** A torrent file exists that has the same name as the file of interest. This file is found by searching the system for the file of interest with `.torrent` appended to the end of the filename. If a file with this extension is found, it is highly likely that the source of the file of interest is a torrent file.
- **fwiredatematch:** FrostWire was used on the day that the file first arrived on the system. This is determined by searching for

references to `FrostWire.exe` under the `UserAssist` key in the registry. An entry is created whenever FrostWire is used. All updates to this key can be seen in the timeline created by running `log2timeline` on the `NTUSER.dat` hive.

4. Experimental Results

The effectiveness of the tool in constructing the provenance of files is demonstrated via six use cases. For each use case, a user conducted a different interaction with a file of interest. After the interaction, the computer media was imaged and analyzed.

Each use case was performed on a computer running Windows 7 Service Pack 2 with Google Chrome, Firefox, FrostWire and Skype installed. The following six use cases were evaluated:

- **Use Case 1:** A user logged on, created a Word document and then logged off. Another user then logged in and edited the Word document.
- **Use Case 2:** A Word document was transferred to the system via USB. A user then edited the Word document.
- **Use Case 3:** A user called someone using Skype and received a Word document from the called party. The file was then moved.
- **Use Case 4:** A user torrented a Word document using FrostWire and then edited the Word document.
- **Use Case 5:** A user downloaded a Word document using Chrome and then edited the Word document.
- **Use Case 6:** A user downloaded a Word document using Internet Explorer and then edited the Word document.

The following flags were set in Use Case 1:

- **difmod:** The file creator and modifier are different.
- **localuser:** The file is in one or more users' recent documents.
- **time_check:** Relevant timeline entries exist that refer to the file within 30 minutes of its creation.
- **systemuser:** The file creator has the same username as a user on the system.
- **word_create_day:** Microsoft Word was used on the system on the creation date of the file.

- **word_appear_day:** Microsoft Word was used on the first day that the file was seen on the system.
- **editing:** The file was possibly edited on the system.

From these flags, a digital forensic investigator can determine that the file was edited and that the editing most likely occurred locally. This is based on the `editing` and `difmod` flags and supported by the `localuser` flag. In addition, the `time_check`, `systemuser`, `localuser` and `word_create_day` flags show that the file was created locally.

The following flags were identified in Use Case 2:

- **usbdatematch:** A removable disk was used on the same day that the file first arrived on the system.
- **timelinerelevant_removable_disk_usage:** A timeline entry references the file of interest as well as a USB device.
- **word_appear_day:** Microsoft Word was used on the first day that the file was seen on the system.
- **difmod:** The file creator and modifier are different.
- **samedaylogin:** The users who last logged in on the same day that the file first arrived on the system are listed.
- **time_check:** Relevant timeline entries exist that refer to the file within 30 minutes of its creation.

The `usbdatematch` and `timelinerelevant_removable_disk_usage` flags indicate that the file could have originated from a USB device. The `word_appear_day` and `difmod` flags show that the file was most likely modified using Microsoft Word after its arrival. The `samedaylogin` flag implies that all the users who logged in that day would be listed, helping narrow down the user who connected the USB device to the system. The `time_check` flag was active due to testing, because the file was quickly transferred after creation for the purposes of this use case; therefore, the presence of this flag can be ignored.

The following flags were identified in Use Case 3:

- **skypedatematch:** Skype was used on the same day that the file first arrived on the system.
- **skype30min:** Skype was used within 30 minutes of the file first being seen on the system.
- **samedaylogin:** The users who last logged in on the same day that the file first arrived on the system are listed.

- **movement:** The file was likely moved in the system file structure.

The `skypedatematch` and `skype30min` flags indicate that Skype was used within 30 minutes of the file's arrival. The `samedaylogin` lists the users who were logged in on the date the file arrived, helping narrow down the user who allowed the file to arrive on the system. The `movement` flag shows that the file may have been moved because the `ctime` and `mtime` are different. The `ctime` and `mtime` values are reported for verification.

The following flags were identified in Use Case 4:

- **torrentfile:** A torrent file exists that has the same name as the file of interest.
- **fwiredatematch:** FrostWire was used on the day that the file was first seen on the system.
- **difmod:** The file creator and modifier are different.
- **editing:** The file was possibly edited on the system.
- **samedaylogin:** The users who last logged in on the same day that the file first arrived on system are listed.
- **impossiblelocal:** The operating system was installed on the system after the file was created.
- **time_check:** Relevant timeline entries exist that refer to the file within 30 minutes of its creation.
- **word_appear_day:** Microsoft Word was used on the first day that the file was seen on the system.

The `torrentfile` and `fwiredatematch` flags indicate a possible torrent source. The combination of the `difmod`, `editing`, `time_check` and `word_appear_day` flags indicate that the file was edited locally with high likelihood. It is unlikely that any of these flags was activated by file creation because of the `impossiblelocal` flag. The `impossiblelocal` flag also indicates the creation date of the file is earlier than that of the operating system. This dramatically decreases the likelihood of local file creation.

The following flags were identified in Use Case 5 and Use Case 6:

- **relevant_chrome_visits:** Chrome was used within ± 2 hours of the file's arrival.
- **relevant_ie_visits:** Internet Explorer was used within ± 2 hours of the file's arrival.

- **difmod:** The file creator and modifier are different.
- **editing:** The file was possibly edited on the system.
- **samedaylogin:** The users who last logged in on the same day that the file first arrived on system are listed.
- **impossiblelocal:** The operating system was installed on the system after the file was created.
- **time_check:** Relevant timeline entries exist that refer to the file of interest within 30 minutes of its creation.

Use Cases 5 and 6 have similar results because they both involve browser history parsing. The `relevant_chrome_visits` and `relevant_ie_visits` flags are both set, which means the tool lists all the web pages visited within a four-hour period. This provides insight into the source of the file, especially because the file's source web page contains the name of the file in the download mirror. The tool presents information in a readable single-line format, enabling a digital forensic investigator to parse the results easily. The usual login (`samedaylogin`) and editing (`difmod` and `editing`) flags are also active, showing that the file was modified locally.

5. Conclusions

This research demonstrates that it is possible to automatically correlate factors related to file provenance. The factors are of great value to digital forensic investigators who seek to determine the origins and activities of files of interest. Typically, an investigator would have to manually mount the image, run various tools and analyze the results in order to determine file provenance. The Provenance Collection Tool presented in this chapter could shave hours off investigations by revealing correlations that would enable digital forensic investigators to quickly focus their attention on more relevant factors.

There are many other checks that, if incorporated, could greatly enhance the functionality and utility of the tool. For example, many different torrent sources exist apart from FrostWire. While the Provenance Collection Tool focuses on FrostWire to show how torrent detection can occur, it is important that it should account for other torrent tools. The same is true for browsers, starting with the implementation of Firefox history checks. The timespans used by the tool to determine correlations are often arbitrary; therefore, the approach can benefit from a large-scale analysis of user activity that would enable the timespans to be narrowed or broadened to render them more effective.

References

[1] F. Buchholz and C. Falk, Design and implementation of Zeitline: A forensic timeline, *Digital Investigation*, vol. 6(S), pp. S78–S87, 2005.

[2] H. Carvey, *Windows Registry Forensics: Advanced Digital Forensic Analysis of the Windows Registry*, Syngress, Cambridge, Massachusetts, 2016.

[3] A. Case, A. Cristina, L. Marziale, G. Richard and V. Roussev, FACE: Automated digital evidence discovery and correlation, *Digital Investigation*, vol. 5(S), pp. S65–S75, 2008.

[4] K. Gudjonsson, Mastering the super timeline with `log2timeline`, InfoSec Reading Room, SANS Institute, Bethesda, Maryland (`www.sans.org/reading-room/whitepapers/logging/mastering-super-timeline-log2timeline-33438`), 2010.

[5] C. Hargreaves and J. Patterson, An automated timeline reconstruction approach for digital forensic investigations, *Digital Investigations*, vol. 9(S), pp. S69–S79, 2012.

[6] P. Harvey, ExifTool (`www.sno.phy.queensu.ca/~phil/exiftool`), 2017.

[7] C. Jensen, H. Lonsdale, E. Wynn, J. Cao, M. Slater and T. Dietterich, The life and times of files and information: A study of desktop provenance, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 767–776, 2010.

[8] G. Lenik, I'm your MAC(b) daddy, presented at *DEF CON 19*, 2011.

[9] D. Margo and R. Smogor, Using provenance to extract semantic file attributes, *Proceedings of the Second Conference on Theory and Practice of Provenance*, 2010.

[10] K. Muniswamy-Reddy, D. Holland, U. Braun and M. Seltzer, Provenance-aware storage systems, *Proceedings of the USENIX Annual Technical Conference*, 2006.

[11] NirSoft, IEHistoryView v1.70 (`www.nirsoft.net/utils/iehv.html`), 2011.

[12] NirSoft, ChromeHistoryView v1.30 (`www.nirsoft.net/utils/chrome_history_view.html`), 2017.

[13] B. Shavers, RegRipper (`brettshavers.cc/index.php/brettsblog/entry/regripper`), 2015.

[14] S. Sultana and E. Bertino, A file provenance system, *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, pp. 153–156, 2013.

[15] E. Zadok and I. Badulescu, A stackable filesystem interface for Linux, *Proceedings of the LinuxExpo Conference*, pp. 141–151, 1999.