Chapter 18

A TOOL FOR EXTRACTING STATIC AND VOLATILE FORENSIC ARTIFACTS OF WINDOWS 8.x APPS

Shariq Murtuza, Robin Verma, Jayaprakash Govindaraj and Gaurav Gupta

Abstract Microsoft Windows 8 introduced lightweight sandboxed applications called "apps" that provide a full range of functionality on top of touch-enabled displays. Apps offer a wide range of functionality, including media editing, file sharing, Internet surfing, cloud service usage, online social media activities and audio/video streaming for the Windows 8 and 8.1 operating systems. The use of these apps produces much more forensically-relevant information compared with conventional application programs. This chapter describes MetroExtractor, a tool that gathers static and volatile forensic artifacts produced by Windows apps. The volatile artifacts are extracted from the hibernation and swap files available on storage media. MetroExtractor creates a timeline of user activities and the associated data based on the collected artifacts. The tool appears to be the first implementation for extracting forensically-sound static and volatile Windows 8 app artifacts from a system hard disk.

Keywords: Windows forensics, Windows Metro apps, forensic timelines

1. Introduction

Microsoft Windows dominates the world's desktop operating system market with a 91% share as of 2014 [15]. Windows 8 (launched in October 2012) and Windows 8.1 (launched in October 2013) have market shares of 6.29% and 6.35% respectively. During the initial period after the launch of Windows 8, there was not much migration to the operating system. However, sales of Windows 8 received a push when touch-screen laptops and monitors became available. Windows 8.1 brought enhanced

G. Peterson, S. Shenoi (Eds.): Advances in Digital Forensics XI, IFIP AICT 462, pp. 305–320, 2015.
DOI: 10.1007/978-3-319-24123-4_18

user interaction and better software integration, which further increased sales. Although, Windows 7 still has about 50% market share, the situation is expected to change in 2015 when Microsoft terminates its mainstream support [10], encouraging users to upgrade to Windows 8 or 8.1 (Windows 8.x).

Windows 8 introduced lightweight application programs called "apps" that work across a variety of computing devices, including desktops, laptops, tablets and smartphones; Microsoft chose touch screens to be the default user interaction medium for all the devices. The apps are more task-oriented compared with full-fledged application programs. For example, there are dedicated apps for checking news and weather, making restaurant reservations, managing finances, accessing online social networks, shopping online and logging health and fitness information. The use of these apps produces much more forensically-relevant information than full-fledged application programs. From the digital forensics perspective, Windows 8.x is similar to Windows 7 in that most forensic artifact extraction methods for application program data that work on Windows 7 can also be used on Windows 8.x. However, the apps, which are exclusive to Windows 8.x, are responsible for the primary difference between Windows 8.x and Windows 7 forensics.

The apps that come bundled with the operating system are called native apps; the remaining apps are downloaded from the Windows app store by the user. The static artifacts of an app mainly include its data and associated metadata stored on the relatively static hard disk drive. These artifacts are available in the local folders and related registry entries of the app. The volatile artifacts of an app include the RAM space allocated to it in physical memory. The hibernation file contains volatile artifacts of apps, desktop applications and other processes in the form of a compressed and encoded dump of the physical memory that contains their most recent running states. The RAM space of an app that is written to the hibernation file contains valuable volatile forensic artifacts that are not present in the app's local folders or registry entries. For example, a Facebook app's RAM space contains news feeds, comments, notifications and likes that cannot be extracted via static analysis. The RAM slack space in the hibernation file can also have archival artifacts pertaining to apps. Similarly, the swap file in Windows 8.x may also contain volatile artifacts related to apps; however, this is limited by the physical memory size and the frequency of the RAM freeing up process. The private data of running apps is flushed to the swap file during a memory crunch [9], which transfers the volatile contents of the apps to the hard disk. Small physical memory size tends to increase the

frequency of RAM freeing up and, consequently, the chances of finding forensic artifacts related to apps.

This chapter describes the design and implementation of the MetroExtractor tool that collects static and volatile artifacts of Windows 8.x apps. The static artifacts are extracted from the hard disk drive and volatile artifacts from the hibernation and swap files present on a Windows 8.x installation disk. The MetroExtractor tool also creates a combined timeline for the static and volatile artifacts collected for a particular app. The tool was tested on the Facebook and Twitter social network apps and the Dropbox and OneDrive cloud data storage apps. The integrity of the results was verified by comparing the findings with a live RAM capture made at the same time. A survey of the literature indicates that MetroExtractor is the first tool capable of extracting forensically-sound static and volatile Windows 8.x app artifacts from a system hard disk.

2. Related Work

This section briefly discusses Windows 8 forensics and the forensic analysis of apps.

2.1 Windows 8 Forensics

This work focuses on the collection of forensically-relevant static and volatile Windows 8.x app artifacts from a hard disk. Johnson [8] has investigated the feature of Windows 8.0 and subsequent versions that helps the users to easily refresh or reset their workstations. He has shown that a user can use these options to restore the original system settings by keeping the user files, removing everything else and reinstalling Windows. However, he did not consider the extraction of volatile data from the hard disk drive.

Thompson [19] has performed similar work, primarily analyzing the app folder structure and the operation of basic apps. His research also investigated the artifacts generated by apps. This work takes a step further by incorporating Thompson's findings in the analysis of app artifacts found in the hibernation and swap files. Thompson also examined Windows 8 registry files in order to collect artifacts. However, the work was done using Windows 8 Consumer Preview and, thus, the results may not be directly applicable to Windows 8.1

Dija et al. [3] have investigated the extraction of volatile artifacts from the hibernation file. Their work concentrates on application programs that can be searched for in the hibernation file; however, it does not consider Windows 8.x apps and the associated page and swap files. Mr-

dovic and Huseinovic [13] have recovered encryption keys from a physical memory dump with a hibernation file as the source. Gupta and Mehte [5] have conducted similar work on the Sandboxie application in a non-encrypted system.

2.2 Forensic Analysis of Apps

This work analyzes some of the common Windows 8.x apps so that correlation techniques can be used to obtain data about online social network activities related to a particular user and/or account. The analysis focuses on the cache of the targeted app and its local information from the corresponding SQLite 3 database. The hibernation, page and swap files are also examined to gather physical memory remnants of the app in the event that the local artifacts of the app were tampered with or deleted.

Iqbal et al. [7] have investigated the Surface tablet with a standard Windows 8 RT installation. However, their work is limited to Windows 8 RT and covers only the Surface tablet, which, compared with Windows 8.x, is restricted in terms of its hardware and operating system functionality. Iqbal et al. have also discussed techniques for obtaining RAM dumps from Windows 7 phones.

Carvey [2] has described the procedures that should be followed during an incident response involving a Windows 8.0 device. He has also identified Windows 8 artifacts that can be extracted. Additionally, he stresses the importance of the hibernation file and the fact that malware and other data can remain resident in the file even after anti-virus software and other temporary space cleaners have been employed.

Quick and Choo [16–18] discuss methods for extracting artifacts related to the use of SkyDrive, Dropbox and Google Drive. The remnants gathered include user-related information and files. Quick and Choo [17] have also analyzed RAM captures and local sources for the Dropbox application, but they did not consider the fact that the hibernation file may contain sensitive data related to the application. Similarly, Wong et al. [20] have focused on finding Facebook artifacts from web browsers and RAM dumps. Beverly et al. [1] have proposed a methodology for extracting network packet data from a hibernation file and using the data in network forensic investigations.

3. Background

This section discusses the hibernation and swap files, and static and volatile artifacts in Windows 8.x systems.

3.1 Hibernation File

Windows 8.x apps are lightweight programs that run on limited resources while providing well-defined functionality to users. User activity information and other data collected by an app is lucrative from a forensic perspective. For example, a map app typically retains all the user queries, including the places searched, routes explored and user location (if a laptop was used). The apps store their data and activity information in their respective local folders and registry entries on the hard disk. However, some of the information is not written back to the hard disk and remains in the app's RAM space when it executes. For example, the Facebook app's RAM space contains news feeds, comments and likes that are not written back to the app's local folders on disk. However, the availability of this extra information in RAM space depends on the type of app and its design.

Experiments were conducted on the hibernation file (`hiberfil.sys`), a large file about the size of the physical memory of the system. The file is overwritten by the RAM contents. At the time of writing the RAM contents to the file, the operating system compresses and encodes the data. Windows 8.x uses the Xpress algorithm for compression and the Huffman and LZ algorithms for encoding [6]. This speeds up the process of writing to and reading from the hibernation file during the shutdown and restart operations. The encoded content of the hibernation file can be converted back to a RAM dump (i.e., in a data format exactly similar to the RAM) using the MoonSols tool [11]. Although Volatility can be used to convert an encoded hibernation file to a RAM dump, MoonSols was found to produce an error-free output at a faster rate. A hibernation file also contains archival data in slack space that is the result of previous writes. The amount of archival data present in a hibernation file depends on the size of the file and the frequency of writes. A general rule of thumb is that the larger the hibernation file, the greater the chances of finding app artifacts related to earlier sessions.

3.2 Swap File

According to the official MSDN blog [9], the runtime design of Windows 8 enables it to suspend an app when it is minimized by a user. The suspended apps do not use the processor as soon they are not in active use. When Windows 8 Metro apps are suspended, the RAM space allocated to them remains untouched. When there is a memory crunch, the operating system flushes all the memory of the suspended apps to a swap file. The memory dump stored in the swap file can be brought back to the RAM later without terminating the apps [9]. Morrison [12]

states that the mechanism of writing the private RAM space of a suspended app to the swap file on disk until it resumes is analogous to the hibernation of an app. Note that the swap file contains the flushed-out private memory dump of an app during a memory crunch, so it does not necessarily hold the last used instance of the app. Hence, the swap file generally provides a random memory snapshot of an app.

3.3 Static vs. Volatile Artifacts

Although the static artifacts of an app on disk give useful insights into user activity and data, the volatile artifacts from the hibernation and swap files are also required to completely reconstruct the sequence of events in a digital forensic case. The importance of collecting static and volatile artifacts of Windows 8.x apps is clarified using two example scenarios. The two scenarios assume that the hibernation file on the system disk cannot be touched by the user because it is inaccessible to all user level apps, applications and processes on a running Windows 8.x system.

- **Unintentional Deletion of Forensic Artifacts:** Assume that a user installs a Windows 8.x app that clears the cache of apps that are already installed on the system. The user has no intention of modifying any evidence related to the pre-installed apps; instead, the user merely wishes to free up some storage space. In this case, the static artifacts are deleted from the system, but can be recovered using forensic tools. However, there is a good chance they are overwritten over time and are unrecoverable. Volatile artifacts in the hibernation and swap files may retain the same archival data that was deleted from the static collection. Additionally, certain forensic artifacts are only present in the volatile space. Therefore, collecting volatile artifacts from the hibernation and swap files is as just important as collecting static artifacts from the disk.

- **Intentional Deletion and Anti-Forensics:** In the second scenario, the user employs anti-forensics to securely delete or tamper with the static artifacts of an app. If the user securely deletes the static artifacts of the app, the artifacts cannot be recovered using conventional forensic tools. If the user tampers with the static artifacts, then there is the possibility that a forensic tool could find traces of the tampering, but there is a good chance that the traces would have been overwritten over time. However, the volatile artifacts in the hibernation and swap files may still have the same archival data that was in the static space.

The two scenarios demonstrate that collecting volatile artifacts from the hibernation and swap files is as worthwhile as collecting artifacts from static space.

4. Experimental Methodology

Windows Metro app static data is present on disk in the form of databases and app registry files. It is important to emphasize that not all the data displayed by the Metro app is stored on disk as database files. Some artifacts are present only in volatile memory. These include, in the case of the Facebook app, the user's news feed, posts made or viewed by the user and the comments made about posts. This data, which is fetched via the Facebook API, is stored in volatile memory and is not reflected on the disk. For example, a user may post a status update on his wall and may decide to delete the post some time later. If such posts were to be reflected on the disk, then errors would exist. Some chat applications require that the recipient should get a message even if it was deleted by the sender. Such activities are, therefore, reflected on the disk. For these reasons, a complete forensic picture may only be available if an investigator considers both static and volatile artifacts. Tools such as Privacy Eraser and other Windows 8 cache cleaners such as Modern UI Apps Cleaner enable users to clean the cache, cookies, Internet history files and other temporary app files from Windows 8 apps folders. This cleaning results in the loss of potential static evidence from the disk.

The MetroExtractor tool described in this chapter gathers static and volatile forensic artifacts of Windows apps. It incorporates a graphical user interface front-end written in Java; its back-end activities are implemented by a collection of shell scripts. MetroExtractor collects static artifacts, arranges them according to their timestamps and presents a timeline of activities. The tool then extracts the hibernation file from the hard disk/`dd` image/user-provided image or E01 file, and converts it to the corresponding RAM dump using the MoonSols memory toolkit. Volatility is then used to extract app-specific memory chunks from the RAM dump. The next step is to extract timestamped data from the chunks. The data is usually present in the form of a JSON file that is received by the (Facebook) app as a response to the API request that it sent. The JSON file is received and parsed by the Facebook app and displayed. The file contains user data such as chats, notifications, image links, textual content and, most importantly, timestamps (Unix epoch timestamps). The JSON data is extracted using regular expressions and is then classified as chats, notifications, posts, etc. on the basis of vari-

able names. The timestamps are then extracted and plotted on a timeline. Finally, the artifacts obtained via the static and volatile analyses are incorporated in the timeline and displayed to the investigator.

The experiments involved the following steps:

1. An app was launched and allowed to load completely and display the data.
2. A RAM dump was created using the MoonSols DumpIt program.
3. The system was made to hibernate.
4. The hibernation file was extracted using a Linux Live CD.

Figure 1 presents the experimental methodology. A RAM dump was obtained just before the system was made to hibernate. This was done to ensure that the RAM content and the content stored in the hibernation file would be as similar as possible to the RAM dump. Two different dumps (one from RAM and the other from the hibernation file) were obtained to verify the findings and if the hibernation file could be used for forensic analysis with the same credibility as the RAM dump. The time difference between taking the RAM dump and hibernating the system was minimized. The app specific memory chunks that were extracted from RAM memory were found to have comparable sizes. The difference in sizes (if present) was never more than 100 MB. The size difference was most likely due to the hibernation file containing some RAM slack.

The MoonSols memory toolkit was then used to convert the hibernation file to a RAM dump. The Volatility Framework could also have been used for this task, but experimentation revealed that MoonSols performs faster conversions. Next, Volatility was used to inspect the RAM dump created from the hibernation file. After obtaining the extracted memory, Foremost was used to perform string searches on the memory dump.

As mentioned above, the back-end functionality of the MetroExtractor tool is implemented by a collection of shell scripts. MoonSols is only used to convert the hibernation file to a RAM dump. MetroExtractor runs on a Linux system or a Windows platform with Cygwin installed. It takes as input a `dd` image, `ewf` file or a hard disk. The tool first converts the hibernation file to an equivalent RAM dump using MoonSols. The RAM dump is then processed by MetroExtractor using the Volatility API to extract the process memory space for the required app. The memory chunk of the app is then processed and classified by MetroExtractor as a chat, comment, notification, etc. Note that the Facebook API responses are in a form specific to JSON with variable names (used for activity

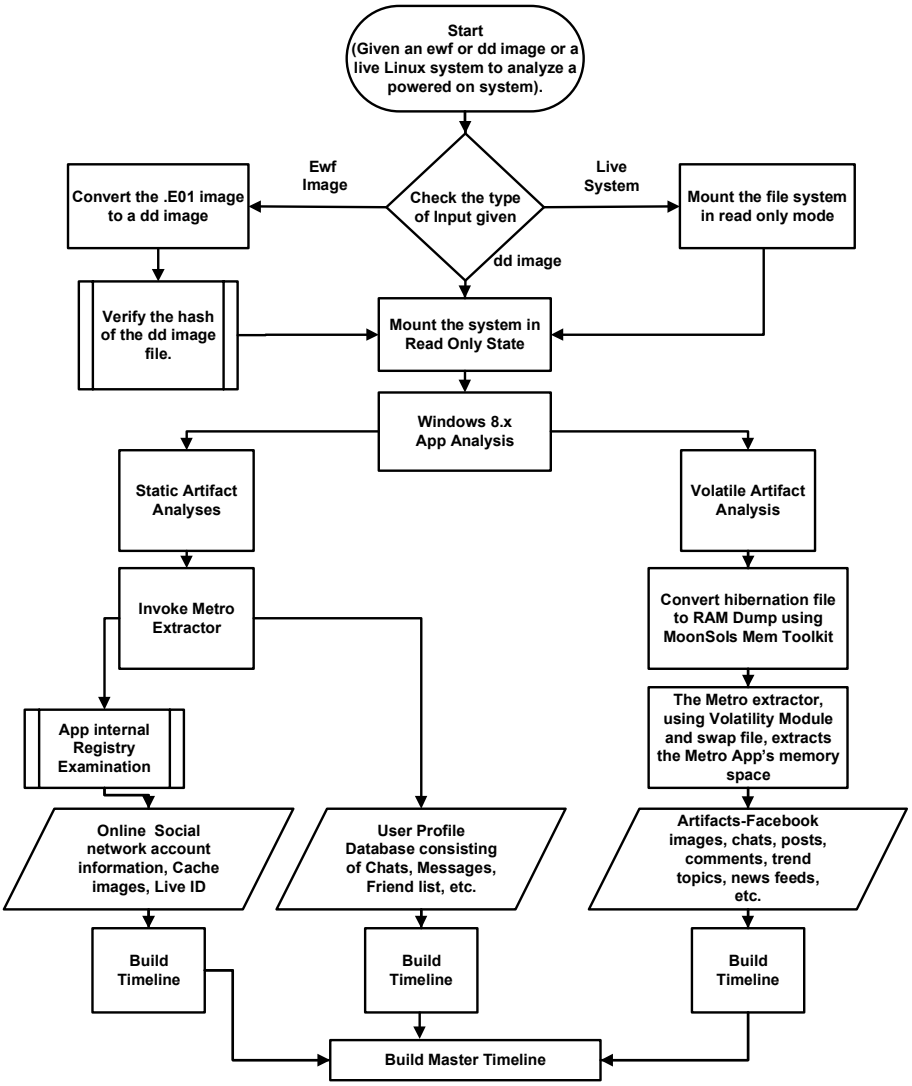

Figure 1. Experimental methodology.

classification) and timestamps. The timestamps are exported into a CSV file along with the corresponding activity.

The MetroExtractor tool also collects static app artifacts from disk. The data is stored in databases from which timestamps are extracted and a timeline plotted. The static and volatile timelines are then merged. Merging the two timelines gives insights about activities that were performed simultaneously, such as commenting and chatting while receiving a notification. The unified timeline representation provides a compre-

hensive picture of user activities. The timeline is displayed by opening the CSV file using spreadsheet software.

- **Facebook App:** The experiment began by launching the Facebook app and allowing it to load all the content. After this, the RAM dump was made and the system was put in hibernation. The RAM dumps were then processed using the Foremost tool.

- **Twitter App:** A similar procedure was followed for the Twitter app. The app was launched and allowed to load the content. After the content was loaded and the app was stable, the RAM dump was made using DumpIt. Following this, the system was put in hibernation, the hibernation file was extracted and the process described above was performed.

- **One Drive App:** By default, documents saved to OneDrive are not downloaded from the cloud; instead, they are downloaded only if requested by users or if users explicitly specify that they should be made available offline. While conducting experiments on OneDrive, it was discovered that the `Skydrive.exe` service always runs in the background in order to synchronize all the data files. Sample files were created and stored in the OneDrive folder. Like the Windows 8 Dropbox app, the Windows 8 Modern app for OneDrive is not designed for synchronization, only for viewing the online stored files. Whenever a file is opened using the app, it is downloaded to temporary storage and then displayed. The files are synchronized by the background service `Skydrive.exe`. Whenever the app was used, it started with the process name `FileManagerexe`. In due course, this process spawned itself as a child process. A similar procedure was followed with the child process.

- **Dropbox App:** The Dropbox Windows 8 app was launched. Dumps were made of the RAM and the hibernation file.

- **OneNote App:** The OneNote Windows 8 app was launched. Dumps were made of the RAM and the hibernation file.

5. Experimental Results

During the experiments, app-specific memory chunks were extracted from the RAM dumps. A similar procedure was followed with the hibernation files after converting them to RAM dumps using the MoonSols memory toolkit. This section presents the experimental results and com-

Table 1. Hibernation file artifacts with live RAM capture (social network apps).

Apps	Hibernation File Data	Live RAM Data	Result
Facebook	Username, profile ID, profile info JSON, profile pic URL, other images, chat messages, Facebook feed status, login email id and trending items.	Username, profile ID, profile info JSON, profile pic URL, other images, chat messages, Facebook feed status, login email id, trending items, news feed, comments, notifications and likes.	The findings were verified to a large extent, the only exceptions were the news feed, comments, notifications and likes.
Twitter	Twitter handle, profile JSON, profile pic URL, tweets at the current instant, embedded pictures, direct messages and email id. The data was present in the form of JSON.	Twitter handle, profile JSON, profile pic URL, tweets at the current instant, embedded pictures, direct messages and email id. The data was present in the form of JSON.	The hibernation file is as good as the live RAM capture.

pares the results obtained from the RAM dumps and the hibernation files.

5.1 Facebook

Table 1 presents the results for the Facebook app. It was possible to extract the images displayed in the user's Facebook feed. Chat messages were found along with user activity and user profile data such as the Facebook profile ID, JSON data pertaining to conversations, timestamps, etc. All the results were verified except for Facebook notifications because the notifications were displayed for a very short time during the live RAM capture. In addition to these artifacts, it was possible to find previous chat messages in the dumps, because the Facebook app was loading them to show the user his chat history. Static artifacts were extracted from the on-disk app databases. These databases contained many details, including the friends list, notifications and chat content.

A timeline was created from the static artifacts, mainly the notification and chat timestamps. The static timeline was merged with the volatile timeline (Figure 2), yielding vital activity clues such as notifications that arrived during chats, when they occurred and the individuals with whom the user was chatting during specific time windows.

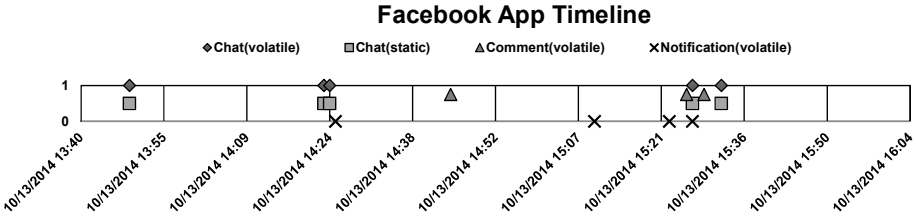

Figure 2. Combined timeline for the Facebook app.

Creating the combined timeline is important because the static and volatile artifacts complement each other. User activities can be verified from the combined timeline. Activities such as chats and friend requests were easily verified from the static artifacts. Many new insights were gained when the static timeline was combined with the volatile timeline. These included the images that were viewed (carved from the dump using Foremost), the verification of the chat history and the comments posted by the user, and the correlation of posts from other users in the user's home page feed.

Table 2. Hibernation file artifacts with live RAM capture (cloud storage apps).

Apps	Hibernation File Data	Live RAM Data	Result
Dropbox	Name list of all files stored in the cloud account, Dropbox user ids with whom files were shared, user login email id, multiple location-based data such as locale and DTS.	All this data was also found in the RAM with no other artifacts being present.	The hibernation file is as good as the live RAM capture.
OneDrive	File list, user name and id (profile and email), SHA1 hash, DTS, sharing level, UID and file sizes.	All this data was also found in the RAM with no other artifacts being present.	The hibernation file is as good as the live RAM capture.

5.2 Other Apps

Other apps, including Twitter, Dropbox and OneDrive, have common artifacts in their static and volatile spaces. Tables 1 and 2 present the results.

In the case of Windows 8 OneNote, the app was started and allowed to synchronize. Following this, the hibernation file and live RAM dump

Table 3. Average time taken by MetroExtractor to process data.

Processor	**System Configuration**	**Operating System**	**Time**
Intel Core2Duo (2.2 GHz)	HDD: 320 GB; RAM: 4 GB	Ubuntu 12.04 LTS (32 bit)	32 min.
Intel Core i5 (2.8 GHz)	HDD: 320 GB; RAM: 4 GB	Ubuntu 12.04 LTS (64 bit)	21 min.

were collected. Microsoft OneNote organizes and stores data in the form of pages called Notebooks. Foremost was able to recover all the images that were pasted in the Notebooks along with the text content. A hex editor revealed an XML structure similar to Office documents. The creator's Live ID and the creation time of the notes were found in the XML structure.

5.3 Performance

The primary time consuming activities of MetroExtractor are the conversion of a hibernation file to the corresponding RAM dump and the extraction of the individual process memory chunks from the RAM dump. Table 3 shows the average time (over ten replications) taken by the tool to process artifacts for a particular app.

Finally, it is useful to compare the performance of `bulk_extractor` [4] and MetroExtractor. The `bulk_extractor` tool works by scanning a disk without parsing the file system structure. MetroExtractor, on the other hand, uses the file system structure to locate active files and extract the corresponding artifacts. Both the tools attempt to find the maximum amount of evidence in the input disk or image. The `bulk_extractor` tool extracts all possible artifacts, including information from other applications. However, the current version of MetroExtractor finds artifacts only for Windows 8 Metro apps. MetroExtractor generates timelines using the timestamps of the collected artifacts whereas `bulk_extractor` simply lists all the evidence present without correlating it or arranging it in chronological order.

6. Conclusions

The MetroExtractor tool described in this chapter gathers forensic artifacts related to Windows 8 and 8.1 Metro apps. Static artifacts from the local app folders and registry entries along with volatile artifacts extracted from the hibernation and swap files are mapped to a single

timeline to provide comprehensive reconstructions of events. The tool was tested on the Facebook and Twitter apps and the OneDrive and Dropbox cloud data storage apps. The integrity of the methodology was verified experimentally by comparing the results from a live RAM capture with those from a hibernation file written to disk at the same time. A survey of the literature indicates that MetroExtractor is the first tool capable of extracting forensically-sound static and volatile Windows 8.x app artifacts from a system disk.

MetroExtractor considers only active hibernation files. Because the Windows 8.x disk optimization options are enabled by default, some portions of a hibernation file may be defragmented. Since the Windows operating system corrupts the beginning of the hibernation file every time the system reboots, a defragmentation operation can cause portions of the corrupted file to be found in the slack space [1].

Future research will focus on developing similar tools for Microsoft Windows phones and tablet devices. Also, research will attempt to analyze synced data that moves between devices.

References

[1] R. Beverly, S. Garfinkel and G. Cardwell, Forensic carving of network packets and associated data structures, *Digital Investigation*, vol. 8(S), pp. S78–S89, 2011.

[2] H. Carvey, *Windows Forensic Analysis Toolkit: Advanced Analysis Techniques for Windows 8*, Syngress, Waltham, Massachusetts, 2014.

[3] S. Dija, T. Deepthi, C. Balan and K. Thomas, Towards retrieving live forensic artifacts in offline forensics, in *Recent Trends in Computer Networks and Distributed Systems Security*, S. Thampi, A. Zomaya, T. Strufe, J. Alcaraz Calero and T. Thomas (Eds.), Springer-Verlag, Berlin Heidelberg, Germany, pp. 225–233, 2012.

[4] S. Garfinkel, Digital media triage with bulk data analysis and `bulk_extractor`, *Computers and Security*, vol. 32, pp. 56–72, 2013.

[5] D. Gupta and B. Mehte, Forensic analysis of Sandboxie artifacts, in *Security in Computing and Communications*, S. Thampi, P. Atrey, C. Fan and G. Martinez Perez (Eds.), Springer-Verlag, Berlin Heidelberg, Germany, pp. 341–352, 2013.

[6] M. Hale Ligh, A. Case, J. Levy and A. Walters, *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux and Mac Memory*, John Wiley and Sons, Indianapolis, Indiana, 2014.

[7] A. Iqbal, H. Al Obaidli, A. Marrington and A. Jones, Windows Surface RT tablet forensics, *Digital Investigation*, vol. 11(S1), pp. S87–S93, 2014.

[8] W. Johnson, Journey into Windows 8 recovery artifacts, *Proceedings of the Conference on Digital Forensics, Security and Law*, pp. 87–100, 2013.

[9] B. Karagounis, Reclaiming Memory from Metro Style Apps, Microsoft Developer Network, Microsoft, Redmond, Washington (`blogs.msdn.com/b/b8/arxchive/2012/04/17/reclaiming-memory-from-metro-style-apps.aspx`), 2012.

[10] Microsoft, Windows Lifecycle Fact Sheet, Redmond, Washington (`windows.microsoft.com/en-us/windows/lifecycle`), 2014.

[11] MoonSols, MoonSols Windows Memory Toolkit, Hong Kong, China (`www.moonsols.com/windows-memory-toolkit`), 2015.

[12] B. Morrison, Windows 8/Windows Server 2012: The New Swap File, Technet, Microsoft, Redmond, Washington (`blogs.technet.com/b/askperf/archive/2012/10/28/windows-8-windows-server-2012-the-new-swap-file.aspx`), 2012.

[13] S. Mrdovic and A. Huseinovic, Forensic analysis of encrypted volumes using hibernation files, *Proceedings of the Nineteenth Telecommunications Forum*, pp. 1277–1280, 2011.

[14] S. Mrdovic, A. Huseinovic and E. Zajko, Combining static and live digital forensic analysis in virtual environments, *Proceedings of the Twenty-Second International Symposium on Information, Communication and Automation Technologies*, 2009.

[15] NetMarketShare, Desktop operating system market share, Irvine, California (`www.netmarketshare.com/operating-system-market-share.aspx?qprid=8\&qpcustomd=0`), 2014.

[16] D. Quick and K. Choo, Digital droplets: Microsoft SkyDrive forensic data remnants, *Future Generation Computer Systems*, vol. 29(6), pp. 1378–1394, 2013.

[17] D. Quick and K. Choo, Dropbox analysis: Data remnants on user machines, *Digital Investigation*, vol. 10(1), pp. 3–18, 2013.

[18] D. Quick and K. Choo, Google Drive: Forensic analysis of data remnants, *Journal of Network and Computer Applications*, vol. 40, pp. 179–193, 2014.

[19] A. Thomson, Windows 8 Forensic Guide, George Washington University, Washington, DC (`propellerheadforensics.files.wordpress.com/2012/05/thomson\_windows-8-forensic-guide2.pdf`), 2012.

[20] K. Wong, A. Lai, J. Yeung, W. Lee and P. Chan, Facebook Forensics, Valkyrie-X Security Research Group (www.fbiic.gov/public/2011/jul/facebook_forensics-finalized.pdf), 2011.