Fall 2012

Methodology and automated metadata extraction from multiple volume shadow copies

Henri Michael van Goethem
James Madison University

Methodology and Automated Metadata Extraction from Multiple Volume Shadow Copies

Henri M. van Goethem

A thesis submitted to the Graduate Faculty of

JAMES MADISON UNIVERSITY

In

Partial Fulfillment of the Requirements

for the degree of

Master of Science

Department of Computer Science

December 2012

Dedication

This work is dedicated to my family for their unwavering commitment and patience throughout my studies for the JMU InfoSec program and this Thesis research effort.

Acknowledgements

I would like to acknowledge the following:

- Tim Leschke for his amazing vision, support, and insight throughout this Thesis research effort as well as for the DC3 Research Mentoring Program sponsorship that initially led to this research concept.

- Rob Lee, Mark McKinnon, Mike Hom, Troy Larson, Harlan Carvey, and the many other extremely talented "digital investigator" trailblazers, who continuously innovate methods of analyzing and unveiling new sources of digital forensics data that reside in the systems we use each and every day. Many thanks for the expert guidance and support these individuals provided throughout this Thesis research. They are the true "shadow warriors."

- Steve Mead and Eric Eifert who provided immeasurable guidance and support throughout the process of completing multiple iterations of peer review and subsequent edits. Here's to "FINAL".doc!

- The JMU InfoSec faculty and staff, who provided outstanding thought leadership, direction, and support throughout my memorable learning experience at JMU. Many thanks for ensuring the InfoSec program focused on proving the theoretical using concrete examples.

Preface

This Thesis research effort discusses the advancement of digital investigative and analysis techniques, resulting in the ability to generate more comprehensive timelines using historical system activity. It is assumed that in conjunction with proper digital investigative techniques, no evidentiary copy of a disk image, volume, etc, would be accessed directly in an investigation, including for the extraction of metadata/data. A suitable working copy should first be made from the evidentiary copy (using appropriate hardware write-blocking technology or approved techniques to safeguard the evidentiary copy). The working copy should then be used for the actual analysis and metadata/data extraction.

Table of Contents

List of Tables

List of Figures

Abstract

Modern day digital forensics investigations rely on timelines [1] as a principal method for normalizing and chronologically categorizing artifacts recovered from computer systems. Timelines provide investigators with a chronological representation of digital evidence so they can depict altered and unaltered digital forensics data in-context to drive conclusions about system events and/or user activities. While investigators rely on many system artifacts such as file system time/date stamps, operating system artifacts, program artifacts, logs, and/or registry artifacts as input for deriving chronological representations, using only the available or most recent version of the artifacts may provide a limited picture of historical changes on a system. For instance, if previous versions of artifacts and/or previous artifact metadata changes are overwritten and/or are not retained on a system, analysis of current versions of artifacts and artifact metadata, such as time/date stamps and operating system/program/registry artifacts, may provide only a limited picture of activities for the system.

Recently, the Microsoft Windows Operating System implemented a backup mechanism that is capable of retaining multiple versions of data storage units for a system, effectively providing a highly-detailed record of system changes. This backup mechanism, the Windows Volume Shadow Copy Service (VSS) [2], exists as a service of modern Microsoft Windows Operating Systems and allows data backups to be performed while applications on a system continue to write to the system's *live* volume(s).[3] This allows a running system to preserve the system's state to backup media at any given point while the system continues to change in real-time. After multiple VSS backups are recorded, digital investigators now have the ability to incorporate multiple versions of a

system's artifacts into a chronological representation, which provides a more comprehensive picture of the system's historical changes.

In order to effectively incorporate VSS backup, or Volume Shadow Copy (VSC), data into a chronological representation, the data must be accessed and extracted in a consistent, repeatable, and, if possible, automated manner.[4] Previous efforts have produced a variety of manual and semi-automated methods for accessing and extracting VSC data in a repeatable manner. These methods are time consuming and often require significant storage resources if dealing with multiple VSCs. The product of this research effort is the advancement of the methodology to automate accessing and extracting directory-tree and file attribute metadata from multiple VSCs of the Windows 7 Operating System. The approach extracts metadata from multiple VSCs and combines it as one conglomerate data set. By capturing the historical changes recorded within VSC metadata, this approach enhances timeline generation. Additionally, it supports other projects which could use the metadata to visualize change-over-time [4] by depicting how the individual metadata and the conglomerate data set changed (or remained unchanged) throughout an arbitrary snapshot of time.

I. Introduction

In the field of digital forensics, timelines [1] have been an invaluable asset for chronologically depicting items of interest during the analysis of digital evidence. They provide digital investigators with an extremely powerful mechanism for organizing, graphically aligning, and analyzing system events and/or user activities both in relation to one-another and to points in time. Timelines also allow investigators to chronologically represent digital forensics data in-context to drive conclusions and ultimately to present the facts surrounding those conclusions to both technically and non-technically oriented audiences. The data used to generate timelines may consist of system artifacts such as file system time/date stamps, operating system artifacts, program artifacts, logs, and/or registry artifacts. The availability of such artifacts for timeline generation/analysis may be affected by a wide variety of influences, including: nefarious user activities, limited auditing and/or audit retention policies, careless or improper systems administration and/or incident response activities, space limitation issues such as log rotation, and other system limitations. As an example, in the course of assessing a potential security incident, the actions of systems administrators may inadvertently alter the access timestamp of certain files. As another example, based on design, storage, and/or retention limitations, some systems may only retain the most recent version of certain files and associated metadata. In all instances, having only the available, or most recent, version of file system artifacts as input for deriving chronological representations may provide a limited picture of historical changes on a system.

In order to offer a solution to this problem area, this research effort first delves into the inherent and largely undocumented backup mechanism of the Windows 7 Operating System, the Windows Volume Shadow Copy Service [2] (VSS), which retains copies of data storage units just prior to changes taking place on a system's *live* volume(s).[3] VSS ultimately captures various file, directory-tree, and time attribute aspects into Volume Shadow Copy (VSC) structures and therefore has the capability to record multiple iterations of change in a chronological context.

Next, an overview and an independent verification and validation (IV&V) of several common methods of accessing VSCs and performing VSC metadata/data extraction is presented. The merits and limitations of each of the existing approaches are reviewed, followed by improvements to advance the automation of VSC metadata extraction in support of timeline analysis. The resulting methodology will ultimately extract metadata from multiple VSCs, which in turn will support the creation of more comprehensive system timelines. As an additional benefit, it will advance efficiencies of the forensics community's open-source capabilities for analyzing VSC contents, as well as potentially support digital forensics projects that visualize change-over-time.[4]

II. Background

Timelines and Time Attributes in Digital Investigations

Change, represented chronologically, is best visualized as a timeline, a linear illustration of important events in the order in which they occurred. By creating timelines of system or user activity and the change(s) created by that activity, digital investigators may depict how various aspects of digital forensic data have been altered. As Daniel points out, computer time artifacts and timelines are critical for validating events and witness claims when an investigation involves an alibi or a set of events that occurred during a specific period in time.[5] Thus, time evidence plays a critical role in the attribution of system behavior and user behavior during specified periods.

Key to the successful ability for a computer to maintain its timekeeping ability are several critical components; namely, the computer's Basic Input/Output System (BIOS), operating system, and file system. The computer's BIOS contains an internal clock which should be established during system setup by calibrating it with a reliable outside time source; subsequently, it will maintain the system's representation of time. A computer's operating system maintains its time in relation to the system BIOS, but also allows for additional customizations based on external influence, such as recognizing various time zones and the respective time changes that occur in each zone at the correct time of the year.

In order for an operating system to record time activity relating to the directory-tree and file attributes, a computer's file system must provide a mechanism for the operating system to record time attributes. Those attributes, which apply to directories and files,

are commonly referred to on Windows-based systems as the *m*odification time (mtime), *a*ccess time (atime), and *c*reation time (ctime), or collectively, MAC times.[6] The modification, access, and creation times record the time at which a file's content is changed in some manner, the time at which the operating system last recorded access to a file, and the time at which a file is first created locally on a file system, respectively.[5] The ability of the computer BIOS, the operating system, and the file system to maintain accurate time and allow for the accurate storage of time attributes provides the basis for our ability to generate timelines of system activities in support of digital investigations. A specialized adaptation that depicts multiple facets of change concurrently, or the *visualization of change-over-time,* is discussed further below.

Visualization of Change-Over-Time

Within the context of digital investigations, Leschke claims depicting change spanned across time is a principal method for digital forensics to answer the supreme question, "what happened?" His research defined change-over-time and its visualization as follows:

> "Time and change share a common quality that is often expressed as the single concept of 'change-over-time.' Because more information can be obtained through vision than through all other senses combined, obtaining information through data visualization presents the greatest bandwidth for human perception. We propose research into using data visualization techniques to enhance the perception of change-over-time as expressed in digital forensic data."[4]

Leschke proposed four (4) approaches to visualizing change-over-time, all of which directly support the role of the digital investigator. These approaches include:

1) visualizing changes to a *directory-tree structure* over time,

2) visualizing changes to *directory-tree content* over time,

3) visualizing changes to *file attributes* over time, and

4) combining the three aforementioned visualization approaches into one conglomerate visualization.[4]

A critical source of digital forensic data that may support the depiction of change-over-time is VSC data; therefore, in addition to supporting the generation of more comprehensive timelines, this research also aims to provide input data which supports the visualization of change-over-time. The background of VSS, methods of accessing VSC metadata/data, and the automation enhancements for extracting VSC metadata/data are discussed in the next subsection and following sections.

Windows Volume Shadow Copy Service

VSS, a service of modern Microsoft Windows Operating Systems that allows incremental system volume backups to be performed while applications on a system continue to write to a system's *live* volume(s), allows a running system to preserve the system's state to backup media at any given point while the system continues to change in real-time. VSS has been in use since the introduction of Windows 2003 Server, is enabled by default on Windows Vista and Windows 7, and is a conglomerate of several underlying technologies that work together to provide incremental backups of data on an arbitrary volume as changes occur.[7] VSS backups occur under three conditions:

1. as part of timed, periodic [8] backups,

2. when new hardware or software installations occur (including Windows Update), and

3. when a user manually initiates a backup via the user interface (UI).[9]

The scope of the service is not limited to specific file types or specified folder locations, as was the case with the predecessor, Restore Points. Rather, except for VSS files and a few temporary files such as paging files, the data corresponding to every file/folder on a system is subject to the incremental backup. This can benefit digital investigations in several ways. First, VSS may record traces of user actions or system changes; therefore, a digital investigator can use VSCs to recover changes and/or establish the timeframe when the system was operational as well as when user activity occurred on the system. Next, VSS may record multiple changes to data which correspond with any arbitrary file(s)/folder(s); thus, digital investigations benefit from having access to a record of changes, or essentially, another form of a "log of change(s)" for the system.

As an arbitrary theft of intellectual property example, let us assume an employee edits multiple company proprietary MS Office documents on his office computer to establish new templates for creating a competitive business. Next, the employee copies the documents to external media and, finally, deletes the original versions from the office computer. If VSS recorded the data changes and file system metadata changes associated with changes to the documents, timeline analysis of the VSCs could greatly enhance a digital investigator's ability to recover as well as link the *before* and *after* versions of the documents.

Digital investigators have begun to rely on VSS as an inherent "logging" and "archival" utility based on its ability to incrementally backup data corresponding to an original

version of a file/folder just prior to the system recording the file/folder's changes to the *live* volume(s). This incremental backup is referred to as copy-on-write technology and for efficiency, occurs at the block level rather than at the logical file level.[10] It is particularly useful as an efficient backup and recovery mechanism for capturing or "snapshotting" *previous versions* of data at a point in time while maintaining the current representation of the data on the *live* volume. Oltean states that VSS backs up the blocks corresponding to a file's data as changes occur and also backs up the blocks corresponding to the Master File Table ($MFT) entry that changes if the file's metadata (size, last modification time, or other attribute) changes.[11] The previous version of the data is viewable and recoverable as long as its data may be rendered in a special representation by overlaying the previous version data or "delta" (from the backup location) against the *live* volume's data. Mullen's example of reading and interpreting a VSC's metadata and data is paraphrased as follows:

> VSS works purely in terms of physical blocks. With regard to reading metadata for a previous version, after reading a VSC's blocks corresponding to the $MFT, the system will interpret them as they existed at the time the backup was created. The system will interpret the overlaid blocks such that each of the recreated $MFT entries point to the content of the clusters as they did at backup time. Additionally, with regard to reading a VSC's blocks corresponding to file/folder data, the system will also interpret the overlaid blocks to "see" the files and folder structure as they were at that time.[12]

Similarly, Oltean's example of reading and interpreting a VSC's metadata and data is paraphrased as follows:

> The read on the VSC works in the following way: let's assume that the user "reads" a file. The read I/O is intercepted by VSS as a sequence of reads for sectors. For the previous versions or "saved" blocks, VSS sends back the "saved" versions of these sectors from the VSC. For the blocks which weren't changed, VSS sends back the current contents from the *live* or "original" volume. In the end, the "read" of the file system for the VSC receives an exact copy of the sectors as they existed at the time of VSC creation.[11]

One benefit the reading and interpreting method offers is that a user may choose to recover the previous version of only one arbitrary file or folder, while not needing to restore an entire volume from backup. Another benefit that makes VSS useful for recovering multiple states of file system data and metadata is that an arbitrary number, *n*, of VSCs (up to the maximum allowed by VSS) may exist at any particular time. Since VSS provides for the visualization and recovery of the previous representation, or version, of a file/folder, Microsoft aptly named the function of the client-side user interface (UI) “Restore Previous Versions.”[10]

The Previous Versions UI is included in all Windows 7 SKUs and allows one to selectively view and restore previous versions of files/folders by right-clicking on a file/folder and selecting the “Restore previous versions” dialogue from the menu. Figure 1 depicts the Previous Versions UI, which is the dialogue an “end user” sees when performing native data restoration “by hand.”

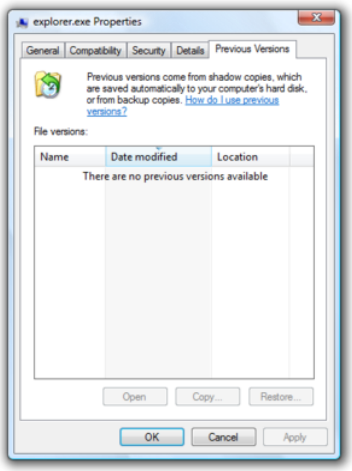


Figure 1: Previous Versions UI

The Previous Versions UI provides the file/folder name, date modified, and the ability to view, copy, or restore an arbitrary object. A user may select any arbitrary file/folder and restore it to a "snapshotted" state. Figure 2 depicts the Previous Versions UI showing several versions of the *FAU.x86* folder that may be restored.

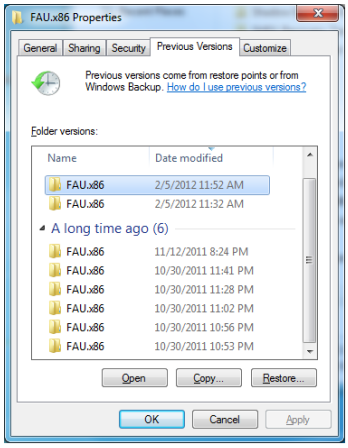


Figure 2: FAU.x86 Folder Versions in the Previous Versions UI

As briefly discussed above, underneath the UI, Microsoft implemented the underlying VSS technology as a block level backup, meaning it backs-up "blocks" of data from the disk versus backing-up the logical files/folders.[13] VSS implements the backup size in 16 kilobyte (KB) blocks.[10] Figure 3, an adaptation from Whitfield's Shadow Warriors

[14], depicts a simplified relationship between a system's logical files, the system's arbitrarily formatted 4KB clusters (NTFS default cluster size under Windows 7), and VSS' 16KB blocks.

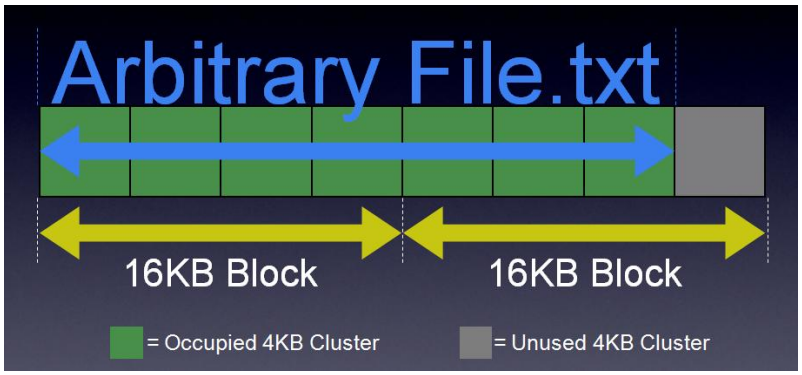


Figure 3: Relationship of Arbitrary File.txt, 4KB clusters, and VSS' 16KB blocks

Time Warp further explains the relationship between a VSC and its data's originating logical volume, which is commonly referred to as the C (or D or other logical) volume.[10] In Time Warp's example, the VSC, or C' volume, is the incremental backup medium for the original "16KB Block" structures as the corresponding, overlying logical file/folder structure changes. Figure 4 depicts this relationship by showing the current state of the "*Arbitrary File.txt*" file, or "*Arbitrary File.txt (Current)*" file, as well as showing the two 16KB blocks that were "preserved" to the VSC, or C' volume. The two blocks from the C' volume may be used in conjunction with the current state to restore the file to its original state, or "*Arbitrary File.txt (Original)*."

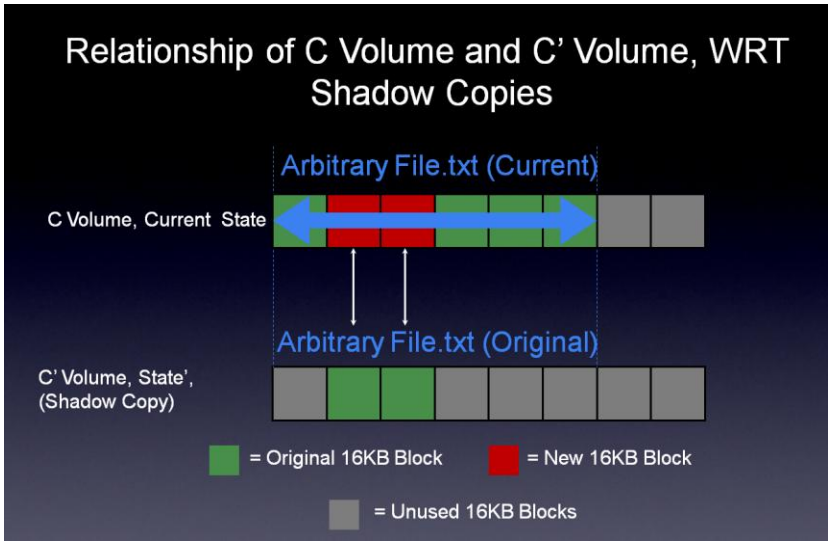


Figure 4: Relationship of Structures and Data on *live* (C) Volume and *VSC* (C') Volume

Understanding this relationship allows digital investigators to better comprehend the relationship between data that is obtained from the *live* volume's current state versus data that is obtained using the *live* volume and one or more VSCs. With regard to timeline analysis, digital investigations benefit from using the metadata/data from one or more VSCs as a complement to the *live* volume, since multiple states of data for the *Arbitrary File.txt* file as well as multiple states of timestamps and other attributes for the file's $MFT entry are available for recovery and subsequent timeline generation.

The creation and management of one or more VSCs is handled by several underlying technologies; in particular, volsnap.sys, the VSS driver, swprv.dll, an intermediary service, and vssvc.exe, the high-level VSS service. Supplementary information regarding the internals of the driver and service functionality is reserved for Appendix A. Additionally, while Figure 4 depicts the relationship between the *live* volume, the two original 16KB blocks of data from the *live* volume, and the VSC, more information is needed to better understand how the VSC concept fits within the Microsoft Windows

Operating System. The following section provides additional insight into the location of VSCs on a real-world system.

Windows Volume Shadow Copies

On a Windows-based NTFS file system and within its logical file structure, VSCs are identified by their GUID filenames and reside in the Windows *System Volume Information* folder, along with other VSS files.[7] VSC structures are "largely self-contained" and are:

> divided into three main parts: the VSS volume header, the VSS catalog, and the VSS stores. … [Note: Metz uses the term "VSS" as "Volume Shadow Snapshot," which is synonymous with this document's use of the term VSC.] The header contains the VSS identifier, which is GUID:{3808876b-c176-4e48-b7ae-04046e6cc752} and the location (byte offset) of the catalog. The catalog contains information about the stores … and the stores contain information about individual 'snapshots.'[15]

The internal structure of the VSCs and other items in the *System Volume Information* folder remain largely unpublished outside of Metz' [15] and Whitfield's [16] work; however, several aspects of the VSS-associated registry structure and VSS processes that contribute to the VSC structure are discussed in further detail in Appendix A. Figure 5 depicts the contents of the *System Volume Information* folder from an arbitrary system, which, with the exception of the "*Windows Backup*" folder, appears consistent across Windows 7 instances.

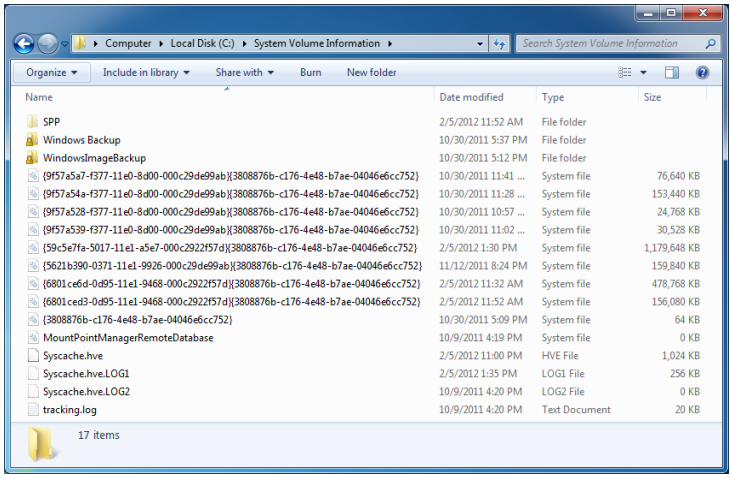


Figure 5: System Volume Information Folder

In order to analyze VSC evidence in support of investigations, digital investigators require the ability to explore the contents inside VSC structures. This can be accomplished in several ways, such as via native Windows rendering or via manual and automated methods, as discussed briefly below and in further detail in the next section.

Rendering VSC Contents

A VSC's contents are not visible as-is to a user in an out-of-the-box Windows 7 implementation, without first interfacing with the VSC. In order to "see" inside the VSCs in the *System Volume Information* folder, the Previous Versions UI must be used to interface with a VSC or the VSC must be accessed via a variety of manual and automated methods (discussed in this section and in additional detail in subsequent sections). Using the VSS driver and services, as well as the Previous Versions UI, Microsoft designed the VSS technology to simulate a disk volume device, providing a static representation of the state of a file/folder/volume at a particular time. To support this functionality, VSCs are queried and a list of snapshot times is returned. As a user selects an arbitrary snapshot, the timestamp associated with the snapshot is used as a reference point for the disk volume's path. The data is portrayed in a manner that allows the VSC's (C', or original)

data to be virtually applied to the *live* (C) volume data in a simulated, or pseudo-volume. This is accomplished via a call to the CreateFile function [17] with the timestamp-laden disk volume's path to mount the simulated volume.[10] Russinovich states the pseudo-volume "path shown will include localhost\C$\<volume label> (<drive>:) (<date>,<time>), which is how Explorer virtualizes the different shadow copies taken."[8] It is marked by the clock with the counter-clockwise green arrow on the left and is depicted inside the red box, below in Figure 6.

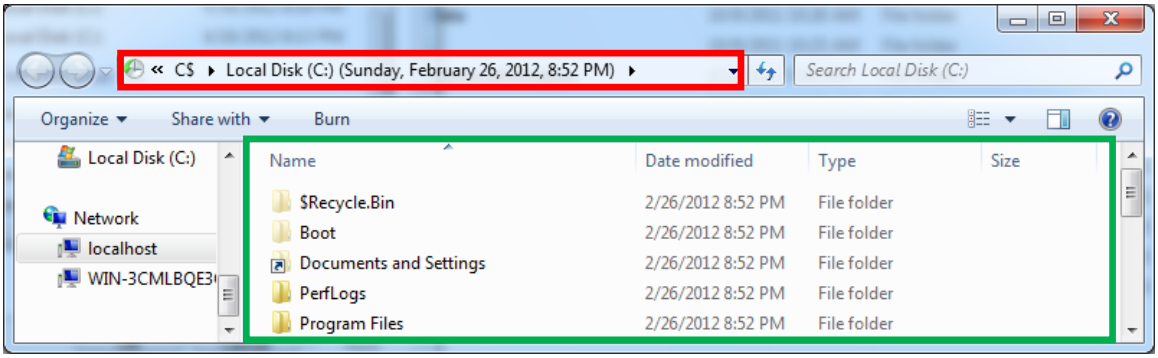


Figure 6: Highlighted timestamp-laden file path of simulated volume

The pseudo-volume representation exists as a read-only structure, which visually represents a logical file and folder structure view of the VSC's contents. Figure 6, inside the green box, depicts the read-only logical structure; in that regard, mounted VSCs are essentially treated as read-only volumes.[10] Of note is that this process works recursively such that multiple VSCs, or incremental backups, may be applied sequentially to render the representation connected to an arbitrary point in time. Figure 7 depicts the recursive nature of the process. It shows how restoration using only the difference blocks of Shadow Copy 3 could take the *live* volume's contents back to the original state for blocks one (A), four (D), six (F), and eight (H), but not for block seven, as only G2 is recoverable using Shadow Copy 3. Additionally, the original state for blocks two (B),

three (C), five (E), and seven (G) are recoverable using Shadow Copies 1 and 2. A restoration recursively using the difference blocks of all three shadow copies is required to take the *live* volume's contents back to the original state for all eight blocks.

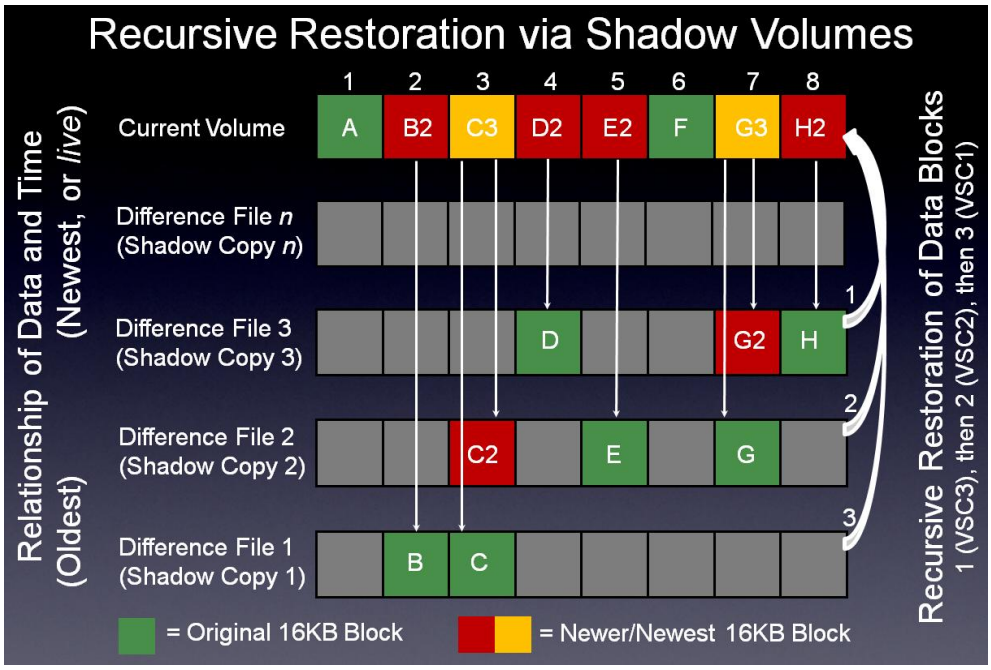


Figure 7: Recursive Restoration via Shadow Volumes

Key to the rendered representation at an arbitrary point in time is the ability to recursively apply arbitrary numbers of incremental changes. Should one or more VSCs (i.e., one or more incremental backups) be missing or corrupted, the rendering and recovery of corresponding previous version data may not be possible. Figure 8, an arbitrary example, depicts this concept.

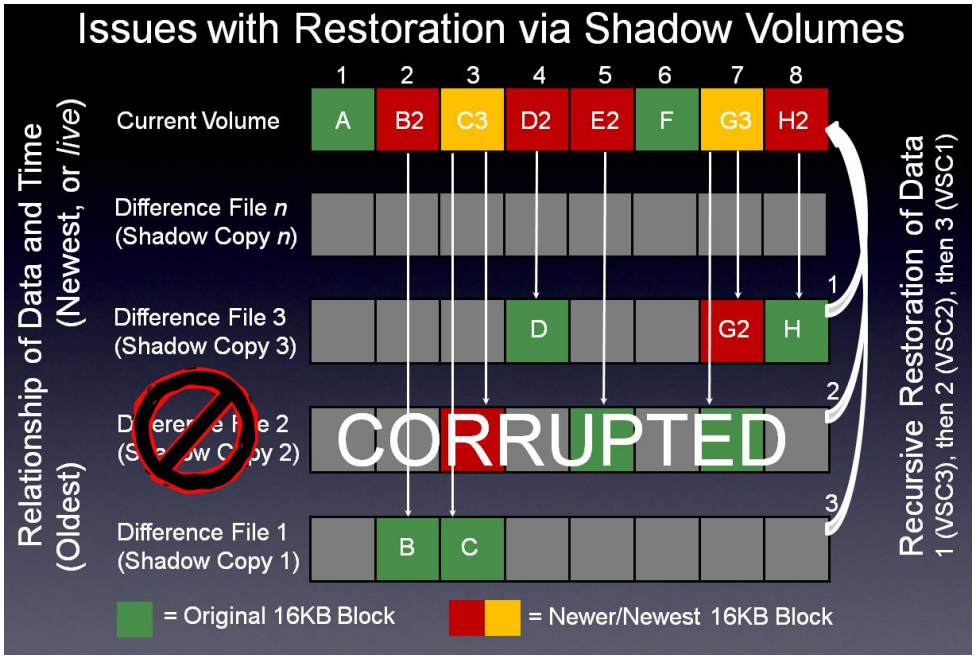


Figure 8: Shadow Volume Restoration Issues caused by Corrupt/Missing Shadow Copy #2

Figure 8 depicts a corrupted and/or missing "Shadow Copy 2" from a set of three shadow copies. Restoration using data from that VSC will not be possible. For example, it would be impossible to restore files/folders which stem from Shadow Copy 2's blocks three (C2), five (E), and seven (G), due to the corruption that has occurred. Expounding on this example, while block seven may be restored to the Shadow Copy 3 block seven (G2) state, due to Shadow Copy 2's corruption, it may never be restored to the (G) state. Conversely, if a healthy VSC only contains a portion of a file's "unchanged" data, the remainder of the file's data must be present on the *live* volume in order for VSS to apply the deltas and successfully read the file in its original state. Thus, if the remainder of the data blocks of the current file have been corrupted or deleted on the *live* volume, recovery of a viable file may not be possible.[9]

Since the aforementioned Shadow Copies consist of incremental blocks associated with "current" files/folders that have since changed on the *live* volume, when an "original"

representation of an object is rendered, the pseudo-volume representation refers to the combination of the *live* volume and a VSC (C' volume). In the instance where no incremental block exists, such as block one (A) and block six (F), the pseudo-volume representation refers only to the *live* volume. In addition to the method for rendering a pseudo-volume, a VSC's contents may also be rendered similarly via other methods, to include mounting the VSC, which is described next.

At a high level, a VSC's contents are visible after a mount point is provided to the operating system and is accessed. Figure 9, an Explorer view, depicts this concept by showing the top-level structure of a manually-mounted VSC (mounted as network share *testshadow20*).

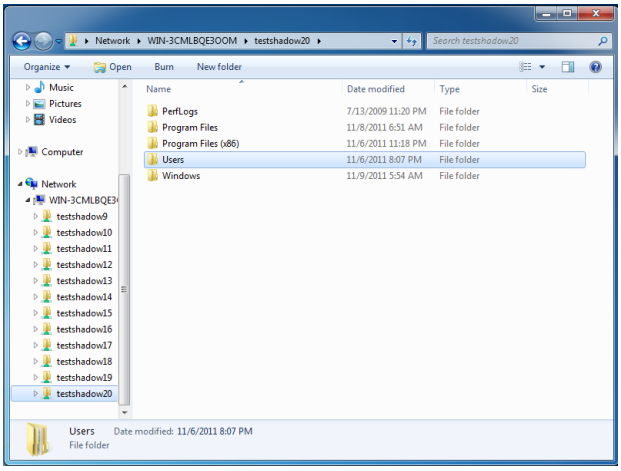


Figure 9: Explorer view of top-level structure within VSC

This method provides digital investigators with the ability to mount and view the contents of any number of VSCs. The ability to mount multiple VSCs in succession allows for the comparison of multiple VSCs' contents in a native Windows Explorer environment.

This section discussed how timelines, time attributes, VSS, and using VSC metadata/data as a complement to the *live* volume are all significant to digital investigations. The next section will provide more context to the approaches digital investigators actively use to

access VSC metadata/data; it will also discuss how the merits and limitations of the approaches are significant to the realm of digital investigations.

III. Digital Investigations Using VSCs

In the area of Digital Investigations, VSCs are useful for restoring a volume to the state in which it existed during the relative time of VSC creation. This is useful for showing how data/metadata (files and folder structure as well as attributes) existed at the time of the backup and allows for recovery of that data/metadata, even though it was changed at a subsequent point in time.[7] More importantly, analysis of VSCs allows digital investigators to interpret how files and folder structure have been altered, enabling them to incorporate multiple versions of a system's artifacts into a chronological representation, or timeline, to provide a more comprehensive picture of the system's historical changes. After multiple VSS backups occur, in order to generate timelines using VSC metadata/data, the data must be accessed and then extracted in a consistent, repeatable, and if possible, automated methodology.[4] Next, the data must be presented in a manner that allows a digital investigator to work with the data -- for example, to identify changes, or lack thereof, over a span of time. The following subsection, Accessing VSC Metadata and Data, describes current manual approaches for accessing and mounting VSCs, and the subsequent subsection, VSC Metadata/data Extraction, describes current manual approaches for extracting VSC contents.

Accessing VSC metadata and data

Investigative work in the digital forensics field by Crabtree [9], Whitfield [14], Lee [19], Carvey [20], Larson [7], "DC1743" [21], and Harrell [22] has produced manual, iterative methodologies for accessing and extracting directory-tree and file attribute metadata/data from VSCs. Several of these methods rely upon mounting/accessing a VSC using a

Windows-based or digital forensics-based utility and then recovering VSC metadata/data using standard Windows-based utilities.

The mount/access methods employ a variety of tools and techniques in order to achieve VSC metadata/data access as well as extraction. While each method brings merits that are beneficial to digital investigations, each also presents obstacles that must be overcome; specifically, the need to:

1. access/mount the VSC(s),
2. access and extract metadata/data from the VSC(s),
3. adopt a format/method that presents the extracted metadata/data to the investigator, and
4. automate the entire process.

The majority of these challenges have been successfully overcome by leading digital investigators. This section provides analysis and discussion of several manual methodologies actively used in support of digital investigations, for the purpose of identifying one that may be automated.

Using Windows Previous Versions

The simplest methodology for manually accessing and extracting VSC metadata/data employs the Windows Previous Versions UI, as depicted in Section II, Figures 1 and 2. The method provides simple point-and-click access to VSC contents and is discussed further in subsection B.1 of Appendix B.

The merit of using the Previous Versions UI approach is the ability to employ a native Windows UI for accurate, easy, and timely data extraction. No additional tools or special methodologies are required; however, this method has several disadvantages, including:

1. The VSC must be accessed via a Windows-based system with Windows Previous Versions UI support (i.e., select versions of Vista, 7, etc.).

2. While an iterative point-and-click approach will extract data from multiple VSCs, the approach does not extract metadata, nor can it be automated in such a fashion as to extract all metadata/data from all VSCs.

The first challenge may be overcome by accessing a VSC using either a surrogate Windows-based system with the Previous Versions UI, or the original system (the latter approach is not recommended due to evidentiary preservation concerns). For example, during IV&V testing, VSCs were accessed using a surrogate system and via VMWare, which was used to help mount the image. The second challenge regarding lack of automation for extracting all metadata/data was not overcome using the Previous Versions UI, but may be overcome using other methods that access VSCs and provide for the extraction of their contents. Therefore, this research effort will analyze and discuss additional methodologies that may overcome this challenge while providing for additional efficiencies. The next method discussed uses *vssadmin* with *mklink* or *net share*.

Using *vssadmin* with *mklink* or *net share*

Digital investigators actively use the Windows *vssadmin* and *mklink* commands to access and mount VSC contents. *Vssadmin.exe* (*vssadmin*), or the Volume Shadow Copy Service administrative command-line tool, is a native Windows 7 command line utility that may be used to display details about VSCs.[23] The *mklink* command executes from within the native Windows 7 command interpreter, cmd.exe. By default, *mklink* creates a symbolic link to a file; however, the *mklink* methodology for accessing VSCs uses the

"/d" argument, forcing *mklink* to create a directory symbolic link instead. This directory symbolic link is used as the mount point for accessing the VSC. This approach provides multiple documented methodologies to mount VSCs, as detailed by Crabtree [9], "DC1743" [21], Carvey [24], Harrell [22], Hargreaves [25], and Oltean [26]. The Windows *vssadmin* tool is also used in combination with the *net share* command to access and mount VSC contents. This approach is used to mount VSCs as Windows-based (network) shares and serves as an alternative for *mklink*'s directory-based access methodology. Figure 9 from the previous section was created using this approach. The use of *vssadmin* with *mklink* or *net share* is described in further detail in subsections B.2 and B.3 of Appendix B.

This methodology is actively used in digital investigations and offers the following merits:

1. It offers the benefit of reliably accessing/mounting one or more VSCs.
2. It may be combined with other techniques in order to extract VSC metadata/data in an automated fashion as well as to maintain the original date and time stamps for extracted data.

While it offers several benefits, this methodology is not without shortcomings:

1. It only provides access to the VSC contents and hence requires additional tools/techniques to extract metadata/data from VSC(s).
2. It provides no automation to select and recover all metadata/data from all VSCs.

3. It also requires additional tools/techniques to store the metadata/data in a format that allows it to be used for timeline generation and/or other visualization purposes.

None of the challenges are overcome without using additional manual and/or automatable methods that access VSCs and provide for the extraction of metadata/data. The next method discussed is restoring and accessing.

Restoring and accessing

To provide another approach for restoring a VSC's contents from the special VSC structure, a methodology was developed to obtain what is commonly referred to as a "disk image,"[27] or forensic snapshot, of a VSC, and then mount that disk image to access its contents. This methodology differs from preceding methods in that it renders a completely new copy of the VSC's contents (combined with the *live* volume) in a "flat" file prior to the VSC mounting process. Essentially, it appears to capture, or image, the entire *live* volume with the VSC's data overlaid to provide the "restored" volume. After imaging is complete, mounting must be accomplished via either the *mklink* or *net share* approach. The restoring and accessing approach is discussed in further detail in subsection B.4 of Appendix B.

This methodology has been used in support of digital investigations and offers the merit of reliably capturing a complete snapshot of one or more VSCs in a forensically-accepted manner. As with the other methodologies presented, this too brings challenges:

1. After mounting the image using either *mklink* or *net share*, this method requires additional tools/techniques to extract metadata/data from VSC(s).

2. It requires additional automation to select and extract all metadata/data from all VSCs.

3. It also requires additional tools/techniques to store the metadata/data in a format that allows it to be used for timeline generation and/or other visualization purposes.

As previously explained, none of the challenges are overcome without using additional manual and/or automatable methods that access VSCs and provide for the extraction of their contents. The next method discussed is parsing VSCs.

Parsing VSCs

A method for accessing VSC metadata/data, without working through Microsoft's VSS application programming interface (API), is to manually parse a VSC and the associated *live* volume, then reverse VSS' incremental backup functionality. This is implemented by applying the "original" VSC blocks in place of the "current" blocks that make-up the *live* volume, and then, reporting the resulting metadata/data. This method, described by ProDiscover as "rebuilding VSC data from the block-level up," works iteratively when multiple VSCs exist, thus reassembly may take a significant amount of time.[28] McKinnon's adaptation of the method is discussed further in subsection B.5 of Appendix B.[29] The approach may be automated into a VSC parsing utility (similar to the ProDiscover and Shadow Analyser utilities, which are discussed in Section IV and subsection C.4 of Appendix C, respectively).

The advantage of this approach is its ability to rebuild the data "from the ground up" without relying on the VSS API.[28] Within the context of this research effort, the challenge of this method is that no open-source tool that utilizes this approach was

available for IV&V during the functional testing and analysis phase. (A promising open-source capability, recently introduced as "alpha" and briefly discussed in the Future Work section, is the *vshadow* project.[15]) The proprietary ProDiscover utility is discussed in Section IV.

In summary, this subsection discussed several manual mechanisms for accessing and mounting VSCs in order to allow for subsequent retrieval of the metadata/data they store. The approaches are summarized in Table 1:

Approach Name	Capability	Limitations
Windows Previous Versions	Point-and-click manual VSC access and data extraction	1. System must have UI 2. Cannot automate
vssadmin with mklink or net share	Manually mount VSC and access data	1. Access only, no extraction 2. Iterative vs. automated 3. Storage format/method
Restoring and accessing	Reliably capture snapshot in a forensically sound manner	1. Access only, no extraction 2. Iterative vs. automated 3. Storage format/method
Parsing VSCs	VSC re-build approach from the ground up	1. No publicly available open-source tool existed during research IV&V phase

Table 1: Accessing VSCs: Approaches, Capabilities, and Limitations Summary

VSC metadata/data extraction

This subsection builds on the manual VSC access and mounting methodologies discussed in the preceding subsection by introducing additional tools and methodologies to extract metadata/data from VSCs. The first approach discussed uses *fls* and *mactime*.

Using *fls* and *mactime* to extract timestamp metadata

The *fls* and *mactime* approach utilizes the filesystem parser and timeline generator utilities to extract timestamp artifacts, which supports the generation of a timeline of

activities.[18] *Fls* and *mactime* are obtained via The Sleuth Kit (TSK).[30] *Fls* “walks through the directory hierarchy [of each partition in a disk image] and outputs a line for each file [and directory] in the file system.”[31] Olsen states that *fls* "operates at the file system layer” and then *mactime* takes *fls* output “and turns it into an ASCII timeline of file activity that's human readable."[32] In order to use this approach, a digital investigator may either use a forensics boot CD containing the *fls* and *mactime* utilities to analyze one or more VSCs or may analyze the VSC(s) on a system that has TSK installed. The approach requires that the digital investigator mount the disk image file containing the VSCs and *live* volume using the Microsoft Windows 7 Computer Management Interface (including the corresponding Disk Manager element) or another utility. The process relies on the Windows disk class driver, volume manager driver, partition manager, I/O manager, CreateFile function, and VSS API to facilitate access to the disk image file as well as the *live* volume and VSCs contained therein.[8] The VSCs are accessed as disk device objects using the device object nomenclature, “\\.\HarddiskVolumeShadowCopy[shadow volume number],” which is similar to the nomenclature used to access standard disk volumes, “\\.\HarddiskVolume[number].”

This methodology does not extract all VSC data, but rather the VSC metadata, such as file and directory names and attribute metadata. Specific actions to accomplish this method are discussed in subsection B.6 of Appendix B.

This methodology is actively used in digital investigations and offers the following merits:

1. It quickly and recursively extracts file and directory names in an automated fashion as well as sorts and formats the metadata based on time stamp information.

2. The *fls* extraction methodology may be scripted/automated for any arbitrary number of VSCs.

3. The flexibility of executing this methodology from an Incident Responder's CD on a running system, in addition to previously discussed approaches, could prove beneficial in exigent circumstances.

Although it provides several benefits, this methodology is not without challenges:

1. It requires additional automation to select and recover all metadata from all VSCs.

2. Based on the combination of the date and time fields within the Date column, it may also require additional tools/techniques to store the metadata in a format that allows it to be used for extensible timeline generation and/or visualization purposes.

The first challenge may be overcome with scripting/automation; however, a platform independent solution for the second is not inherent in this methodology's capabilities. The next approach discussed involves using "specialized" methods.

Using specialized utilities/methods

Carvey discusses the ability to use a variety of specialized forensics utilities, such as RegRipper, a Windows Registry data extraction and correlation tool, to extract metadata from VSCs.[20] After a digital investigator gains access to the VSC, RegRipper may be

used to extract useful data from any arbitrary registry key. The methodology is discussed briefly in subsection B.7 of Appendix B.

This methodology is actively used in digital investigations and offers the following advantages:

1. In addition to the preceding two extraction methodologies, RegRipper further demonstrates that any utility may be used to extract information from VSCs.
2. The extraction methodology may be scripted/automated for any arbitrary number of VSCs.

As with the other methodologies explored, this approach is not without its disadvantages. Challenges are as follows:

1. This methodology in and of itself does not select and recover all metadata/data from all VSCs (the majority of arbitrary utilities would require additional automation to access the VSCs and then select and recover all metadata from all VSCs).
2. It also requires additional automation as well as tools/techniques to store the metadata in a format that allows it to be used for timeline generation and/or other visualization purposes.

A portion of the first challenge could potentially be overcome with scripting/automation; however, a solution for the second is not inherent in this methodology's capabilities. Additional approaches will be discussed and analyzed in order to overcome these challenges.

After presenting several manual approaches for accessing and mounting VSCs in the previous subsection, this subsection discussed two manual approaches for extracting VSC metadata/data. Table 2 summarizes both:

Approach Name	Capabilities	Limitations
fls and mactime	Extract VSC metadata quickly	1. Iterative vs. automated 2. Storage format/method
Specialized utilities/methods	Extract VSC metadata quickly	1. Iterative vs. automated 2. Storage format/method

Table 2: Extracting VSC Contents: Approaches, Capabilities, and Limitations Summary

This section also provided the rationale for the use of VSC metadata/data in support of digital investigations. The next section, Achieving Automation for VSC Metadata/Data Enhancements, builds upon this information by using scripting and coding of open-source and commercial products for more efficient and somewhat automated VSC access and metadata/data extraction.

IV. Achieving Automation for VSC Metadata/Data Extraction

The previous section discussed several of the commonly used manual techniques for accessing and mounting VSCs; it also discussed several of the commonly used methods for extracting metadata/data from VSCs. In order to tackle the scale of addressing metadata/data from multiple VSCs and gain additional efficiencies in support of digital investigations, methods of enhancing/automating the access and extraction approaches are explored further in this section. Four actively used open-source approaches and two commercial utilities will be analyzed to determine whether an automated methodology exists that best addresses the challenges previously identified by this research effort. The open source methodologies and tools are:

1. scripting manual tools [22],
2. Robocopy [33],
3. LogParser [34], and
4. ShadowCopy.[35]

The commercial utilities are:

1. Shadow Explorer [36] and
2. ProDiscover.[28]

The first of the four open-source methods, scripting manual tools, provides an "introduction" into automation and efficiencies.

Scripting manual tools

As discussed in Section III, several manual methods may be used to iteratively complete a three-step process of mounting a VSC, extracting its contents, and unmounting the

VSC. Enhancements to this methodology, such as using programming loops, are actively employed by digital investigators, such as Crabtree [9], "DC1743" [21], and Hargreaves [25]. The programming loops approach first mounts a disk image file (of raw (*dd* [37]), Virtual Hard Disk (VHD [38]), or Virtual Machine Disk (VMDK [39]) format) and then uses a *for* loop to iteratively execute either the *mklink* command, thus creating directory symbolic links, or the *net share* command, thus creating a share, for each of the VSCs. After the VSCs are mounted and accessible via the directory symbolic links or shares, their contents are viewable for analysis and extraction using various utilities. When analysis is complete, unmounting, or "cleanup," of the VSCs is performed by using a *for* loop to iteratively execute the *rd* command.

Harrell further automates the *for* loop controlled VSC mount/dismount process using *vssadmin*, *mklink*, and *rd* by encapsulating it within a batch script.[22] Hargreaves also provides a command string, which may be added to a batch file to "mount all Restore Points simultaneously."[25] This method is further discussed in subsection C.1 of Appendix C. Figure 10 depicts the output of this methodology, VSCs that are mounted and viewable as folders within Windows Explorer.

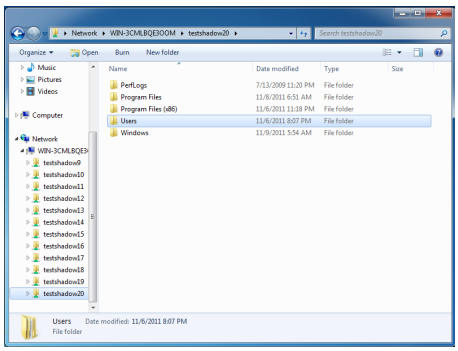


Figure 10: Mounted VSCs, now accessible via Windows shares

This methodology is actively used by digital investigators. Its merits, include:

1. It offers the benefit of reliably accessing/mounting one or more VSCs with *limited automation*.

2. The scripting enhancement to this methodology provides even more automation and, when combined with other techniques, such as those performing metadata extraction, may provide initial steps for *enhanced automation*.

This method also has a few challenges, which are as follows:

1. It requires additional tools/techniques to extract all metadata/data from all VSCs.

2. It also requires additional tools/techniques to store the data in a format that allows it to be used for timeline generation and/or other visualization purposes.

A portion of the first challenge could potentially be overcome with scripting/automation; however, a solution for the second is not inherent in this methodology's capabilities. The next method discussed is *robocopy*.

Using *robocopy*

Microsoft's Robust File Copy for Windows, or *robocopy*, utility allows digital investigators to "copy out folders and files of interest from any notable shadow copies. The process will preserve folder and file paths and timestamps. The key advantages are that it is efficient - both in storage and speed."[40]

Robocopy has received significant interest/use from the digital investigations community, based on its reliability in extracting metadata/data from VSCs while maintaining the original date and time stamps. It also offers the flexibility of extracting all metadata/data from a VSC, or the precision of extracting metadata/data from a single file/folder of

interest within a VSC. The methodology is described in further detail in subsection C.2 of Appendix C.

Validation testing with the *robocopy* methodology produced metadata of the file and folder information from within an arbitrary VSC, or when performed iteratively, multiple VSCs. Figure 11 depicts the output produced using this approach, which first shows the *robocopy* command with arguments, followed by an excerpt of the *robocopy* log file, displaying the fields *Type* (i.e., New File, New Dir, or junction), (<*Number of Files a Directory Contains*>), (<*Size*>) in Bytes, (<*mtime*>), and Path.

```
C:\Windows\system32>robocopy c:\vsc17 d:\Robocopy_test /E /XJ /w:0 /r:0 /log:d:\
robocopytest.log /L /X /V /TS /FP /BYTES /TEE /NJH /NJS

 Log File : d:\robocopytest.log

                  2    c:\vsc17\
       New File          1006280704 2011/11/07 03:37:22  c:\vsc17\hiberfil.sys
       New File          1341710336 2011/11/07 03:37:33  c:\vsc17\pagefile.sys
       junction     -1   c:\vsc17\Documents and Settings\
       New Dir       0   c:\vsc17\$Recycle.Bin\
       New Dir       1   c:\vsc17\$Recycle.Bin\S-1-5-21-1616509852-2306045778-1518815187-1000\
       New File            129 2011/11/07 01:07:42    c:\vsc17\$Recycle.Bin\S-1-5-21-1616509852-2306045778-1518815187-
1000\desktop.ini
       New Dir       0   c:\vsc17\PerfLogs\
       New Dir       0   c:\vsc17\PerfLogs\Admin\
       New Dir       1   c:\vsc17\Program Files\
       New File            174 2009/07/14 04:54:24    c:\vsc17\Program Files\desktop.ini
...
```

Figure 11: Resulting log of *robocopy* methodology

The *robocopy* methodology has merit in that:

1. It extracts VSC metadata/data in an *automated* fashion as well as maintains the original date and time stamps of extracted data.

2. It also offers many execution argument options, providing flexibility for specifying output content and format.

Robocopy suffers from a few shortcomings, which are as follows:

1. While it extracts metadata/data from VSCs, it does not currently capture enhanced file and folder attribute metadata in the log. I.e., this methodology does not capture MAC times information and it also does not capture file and folder attribute information.

2. *Robocopy*'s log is a flat file comprised of text entries; additional tools/techniques are required to process and store the metadata in a format that is conducive to easy retrieval, timeline generation, and/or other visualization purposes.

It appears that neither challenge may be overcome without the addition of one or more third party utilities. The next method discussed is *LogParser*.

Using *LogParser*

The *LogParser* utility is used by digital investigators to export metadata from VSCs into a comma-separated value (CSV) format. The methodology sends the output of the *vssadmin* command to a text file on the analysis system, mounts all VSCs using the *mklink* command, and then determines the contents of each VSC utilizing the *LogParser* utility. *LogParser* grabs the metadata for all the files and folders within each VSC and exports the metadata to CSV format.

Validation testing with these options extracted metadata from an arbitrary VSC, or when performed iteratively, multiple VSCs. Table 3 depicts the first five records of output produced using this approach.

MD5 Hash	Creation Time	LastWriteTime	LastAccessTime	Attributes	Name	Path	Size (Bytes)
	7/14/2009 3:18	11/7/2011 1:07	11/7/2011 1:07	D-SH-----	$Recycle.Bin	C:\VSC17\$Recycle.Bin	0
	7/14/2009 5:08	7/14/2009 5:08	7/14/2009 5:08	D-SH---N-	Documents and Settings	C:\VSC17\Documents and Settings	0
15E2F5A2AB8A534534386CF5E068F950	11/7/2011 6:49	11/7/2011 3:37	11/7/2011 6:49	-ASH-----	pagefile.sys	C:\VSC17\pagefile.sys	1341710336
	7/14/2009 3:20	7/14/2009 3:20	7/14/2009 3:20	D--------	PerfLogs	C:\VSC17\PerfLogs	0

	7/14/2009 3:20	7/14/2009 7:47	7/14/2009 7:47	D---R----	Program Files	C:\VSC17\Program Files	0

Table 3: Resulting records of the LogParser methodology

The output in Table 3 depicts the VSC's contents, to include the MD5 hash value, creation time, last write time, last access time, attributes, name, path, and file size (in bytes). The time may be specified as local time or UTC; UTC was implemented during validation testing. Additional analysis regarding this methodology is provided in subsection C.3 of Appendix C.

The advantages of the *LogParser* approach are as follows:

1. It extracts metadata from VSCs in an *automated* fashion and does so without altering the original date and time stamps of the source files.

2. *LogParser*'s ability to push VSC names to a text file to distinguish input sources, as well as its ability to extract metadata from multiple VSC's into a single file, may allow for additional automation.

3. The *LogParser* utility also offers the *System_TimeStamp()* and *System_UTCOffset()* functions, which are helpful for baselining the time of a digital investigator's analysis system.

The LogParser approach has the following challenges:

1. Since LogParser's automation stems from extracting metadata from iteratively selected VSCs, it requires additional automation to select and recover all metadata from all VSCs.

2. Validation testing produced multiple instances of the following error: "Error retrieving files: Error searching for files in folder <folder>: Access is denied." Escalating privileges to NT Authority\System reduced the error frequency, however, it did not eliminate all instances.

3. This methodology requires additional tools/techniques to extract VSC data such as folder structure and files.

Overall, the *LogParser* methodology offers significant enhancement potential for mounting all VSCs, extracting all metadata from all VSCs, and storing the data in a format that is conducive to future retrieval, timeline generation, and/or other visualization purposes. In order to overcome the disadvantages posed by the *LogParser* methodology, this research effort will discuss the *shadowcopy.py* approach next.

Using *shadowcopy.py*

The most extensible open source VSC metadata/data-extraction methodology analyzed during this research effort was *shadowcopy.py*. *Shadowcopy.py* is a Python programming language script produced by Mike Hom of the Naval Postgraduate School in Monterey, California. Hom used Brian Madden's Python script, ShadowVolume2.py, for accessing VSCs, and then wrote *shadowcopy.py* as the enhancement script for automating VSC parsing.[35]

Shadowcopy.py was written with the intended use of Python version 3.2 libraries and therefore requires the Python interpreter to exist on the digital investigator's examination system. Since *shadowcopy.py* is Python language-based, it offers the flexibility of multi-platform execution. *Shadowcopy.py* requires Administrator-level access rights as well as access to a VHD format disk image converter utility, vhdtool.exe (analysis of the vhdtool.exe method is discussed in Appendix B); however, both conditions are easily met on a digital investigator's examination system.

The *shadowcopy.py* approach offers the following merits:

1. It extracts all accessible data, such as folder structure and files, in an automated fashion.

2. The Python script approach offers complete flexibility of executing multiple third party utilities as well as many execution arguments.

3. *Shadowcopy.py* also deduplicates extracted data based on the MD5 one-way hash value, and produces a report of each extracted file's name, MD5 hash, size, originating machine, VSC, and destination location (extraction directory path). Additionally, if the same file name is encountered, but with a different hash, *shadowcopy.py* will extract both files and rename all versions subsequent to the first with a three digit numerical delimiter.

4. Another merit is this method's ability to distinguish between processing the VSCs of the local (host) system or all non-local (external) VSCs; this provides an automated methodology for a digital investigator to process all evidentiary, or non-examination-system, VSCs.

5. One final, critical *shadowcopy.py* merit is that the Python script is extensible and the Python language affords flexibility for additional automation, which provides outstanding extensibility for modifications and enhancements.

Challenges of the *shadowcopy.py* approach are as follows:

1. *Shadowcopy.py* extracts VSC data, such as directory structure and files, but it does not currently capture all common directory structure, file, and attribute metadata into the report file. For example, it does not record directory structure, MAC times, or file attribute information, which could be critical in helping a digital investigator to determine "items of interest" in the report file. This

provides an incomplete picture, which makes it difficult for a digital investigator to use the report for gathering an initial picture of exactly what changed and when, outside of relying on the obvious filenames and hashes. Additionally, during functional testing, the MAC times of the extracted VSC data, such as folder structure and files, were not restored to their original MAC times from within the VSCs, making the comparison of exactly when things changed or comparison of one item against another even more difficult.

2. *Shadowcopy.py* also does not currently support a seamless method for exporting metadata into a storage format conducive to future retrieval, timeline generation, and/or other visualization purposes. While the existing methodology exports limited metadata to a tab-delimited file, without first importing the metadata into other repositories or storage formats, it is difficult for a digital investigator to review all VSC metadata to determine which data, such as folder structure and files, is of value to the investigation.

3. *Shadowcopy.py* can currently only extract all data and/or the aforementioned limited version of all metadata. Due to the additional storage space required for capturing all exported data, extracting all data prior to conducting a cursory and/or thorough metadata/timeline analysis is not conducive to a digital investigator's analytical efficiency or effectiveness. While the *shadowcopy.py* approach offers some data extraction efficiencies by eliminating duplicate files, the approach could be improved upon by extracting all needed metadata first and then subsequently offering the option to extract all data or only a subset of *investigator-targeted* data.

4. A final challenge, which is identified by the author, is *shadowcopy.py*'s inability to mount VHD format or raw (*dd*) format disk image files in an automated fashion. Currently, the user must manually mount the files external to the *shadowcopy.py* script.

Overall, the *shadowcopy.py* methodology and its extensibility, based on the second and fifth merits listed above, offers significant potential for:

1. mounting all VSCs,
2. extracting metadata from all VSCs, and
3. storing the data in a format that is conducive for future retrieval, timeline generation, and/or other visualization purposes.

In order to identify other potential best of breed capabilities which may enhance the *shadowcopy.py* approach, commercial and open-source GUI utilities will be analyzed next.

Commercial & Open Source GUI Utilities

In addition to the manual and automated approaches for accessing VSCs and extracting VSC metadata/data, several more automated and *visually-enhanced* utilities exist for providing even greater efficiencies to digital investigators. These open source and commercial tools are equipped with GUIs to further automate the review, analysis, and extraction of metadata/data within VSCs. Two easily obtainable utilities, Shadow Explorer and ProDiscover, are discussed in this section.

Using *ShadowExplorer*

ShadowExplorer displays VSC content in an Explorer-like interface and allows digital investigators to export VSC data, such as any file or folder, to an output folder on a

storage medium of choice.[41] *ShadowExplorer's* Windows Explorer-like interface is easy to follow and is depicted below in Figure 12.

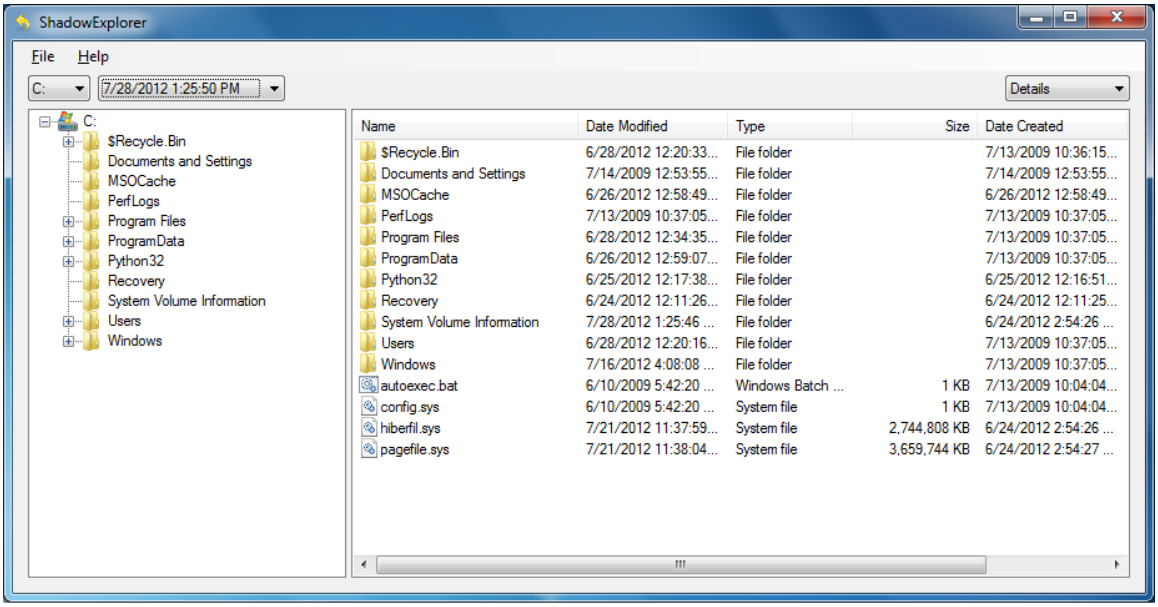


Figure 12: *ShadowExplorer* Interface Depicting a single VSC

The utility allows the user to choose one (note: only one) VSC to view at a time from a list. In addition to viewing a VSC in the Explorer-like interface, an operator may right-click and export any file(s) and/or folder(s) to an export directory of choice.

ShadowExplorer's quick and simple, easy-to-use interface is valuable for gaining a cursory review of VSC contents. The utility offers a single function consisting of a right-click followed by the *Export* option, and performed the function as anticipated during validation testing. Unfortunately, the *ShadowExplorer* utility suffers from several drawbacks, which are as follows:

1. It provides no automation to select and recover all data from all VSCs.

2. It does not offer the ability to extract VSC metadata in a format that allows it to be used for timeline generation and/or other visualization purposes.

3. Based on GUI limitations, the list of available VSCs was only updated once during execution (i.e., there was no "refresh" mechanism to update the VSC list).

4. *ShadowExplorer* did not appear to recognize VSCs from a source other than the native drive upon which the running OS resided (i.e., it did not allow viewing of the VSCs that were mounted via the *diskpart* utility).

The many challenges associated with the *ShadowExplorer* approach allowed us to eliminate it as a candidate for enhancing support to digital investigations involving VSCs. In order to identify other potential best of breed capabilities which solve the problems previously identified, this research effort will discuss one final methodology: using ProDiscover.

Using *ProDiscover*

Technology Pathways produces a family of ProDiscover utilities, including the ProDiscover Incident Response (IR) utility, which provides an "integrated way for investigators to access Volume Shadow Copies from within the digital forensics environment."[28] ProDiscover's documentation explains that the capability is achieved by analyzing and rebuilding a view of VSC data from the block-level up. The utility allows investigators to mount and image VSCs from *live* machines, mount VSCs from any supported image format, and mount VSCs from any directly added "at rest" disk.

ProDiscover IR offers two underlying methods of processing and visualizing recreated VSCs. According to documentation, in the first method, ProDiscover IR parses VSCs. In the second method, ProDiscover IR uses Microsoft's VSS API for processing and visualizing VSCs, which provides for quicker analysis.

Testing confirmed ProDiscover IR offers the ease of a point-and-click methodology to mount any VSC that is accessible via a physical drive or supported drive image. Figure 13 below depicts the ProDiscover IR "Mount Shadow Volume" dialogue box which shows how a VSC and its mount point (volume) are selected. Figure 14 below depicts the ProDiscover IR interface after two VSCs are mounted. After VSCs are mounted, the interface displays them in the same manner as standard logical volumes, showing the directory structure and files as well as attributes of VSC contents. ProDiscover IR allows easy "side-by-side" visual comparison of VSC metadata/data.

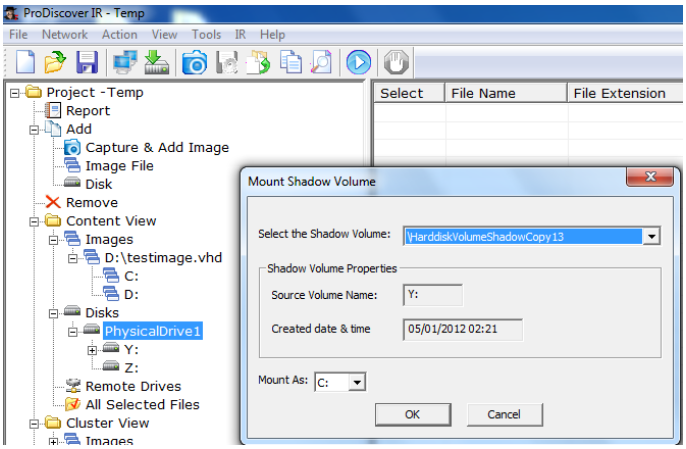


Figure 13: ProDiscover IR Dialogue for Mounting a VSC

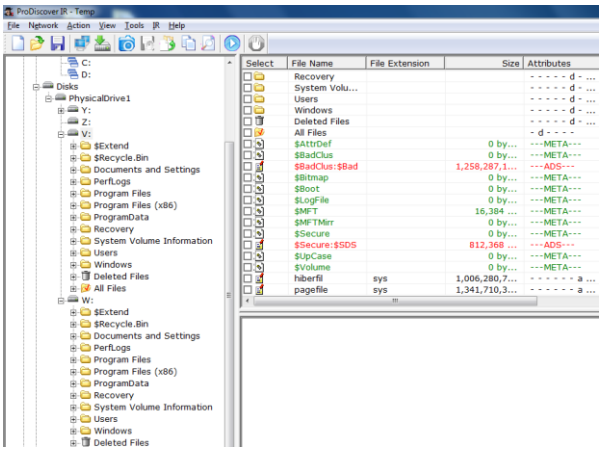


Figure 14: ProDiscover IR Visualization of Two Mounted VSCs

By selecting the "Tools" and "Compare Volumes" options, ProDiscover IR offers the ability to automatically compare the contents of one-to-many VSCs. This method works by computing a hash for every file on every volume selected, which requires significant processing power and time. After this option is executed, ProDiscover IR provides a visual depiction of the contents that have changed and offers the ability to filter results by action (i.e., no filter, added, modified, or deleted) or by file type (i.e., no filter, .txt, .doc, .pdf, .jpg, or .bmp). A report of results is saved to .txt, .xml, or .csv file format. Figures

15 and 16 depict the “Compare Volumes” dialogue box and the results of a simple one-to-one volume (VSC) comparison, showing results filtered by .txt filetype.

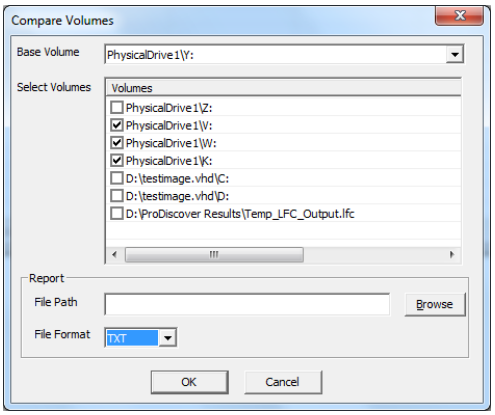


Figure 15: ProDiscover IR Compare Volumes dialogue box

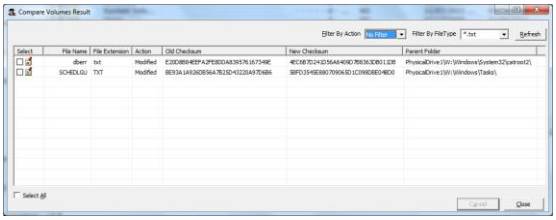


Figure 16: Compare Volumes Result dialogue box filtered by “.txt” filetype

ProDiscover IR also offers the ease of a point-and-click methodology to compare VSC changes via the “Extract Volume Shadow Copies” methodology, which determines changes by examining the $MFT and then comparing timestamps. It extracts differences into a Logical File Collection (LFC) and offers the ability to preserve the directory structure. Since the methodology compares timestamps of VSC contents with the $MFT, it completes one-to-one VSC comparison at a time, but offers faster execution over the “Compare Volumes” method. After the process completes, ProDiscover IR has the capability of adding the LFC to the investigation, thus allowing digital investigators to display the changes between two particular VSCs. Figures 17 and 18 depict the “Extract Volume Shadow Copies” dialogue box and the ProDiscover IR GUI showing two mounted LFCs, respectively.

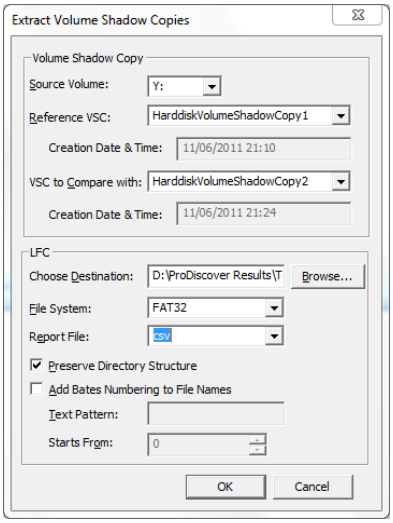


Figure 17: Extract Volume Shadow Copies dialogue box

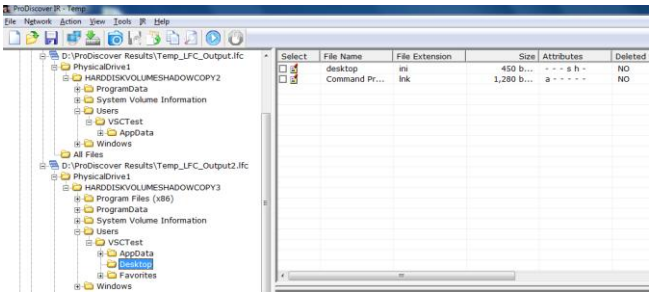


Figure 18: Mounted LFCs Depicting Changes Between VSCs

ProDiscover is actively used by digital investigators and offers the following merits:

1. It provides an easy-to-use GUI, allowing digital investigators to mount one-to-many VSCs (one at a time) via point-and-click methodology.
2. It offers the ability to capture and extract differences in VSC content metadata via the Compare Volumes feature.
3. It can compare VSC data/metadata and visualize differences by selecting "Extract Volume Shadow Copies" and subsequently mounting the resulting LFCs.
4. It requires only minimal point-and-click user interaction to select, recover, and export the differences between selected VSCs for either of the aforementioned methods.
5. It extracts VSC metadata into several exportable formats. The utility performed all functions as anticipated during validation testing.

The ProDiscover utility presented the following disadvantages:

1. The need to sequentially process differences between multiple VSCs was not fully automated/eliminated. For example, one may compare the differences between a VSC and the *live* volume or any two VSCs; however, in order to complete the comparison process sequentially for *n* VSC or *live* volume items from a particular drive or drive image, *n-1* iterations are required.

2. The ProDiscover IR retail cost is approximately $8,000.00 US dollars per license, causing the functionality to be restricted to only those who may obtain a demo version of the utility or afford the license costs.

3. ProDiscover IR's log format is a flat file (.txt, .xml, or .csv format) comprised of text entries, therefore, this utility does not inherently extract and store VSC metadata in a format that is conducive to retrieval, timeline generation, and/or other visualization purposes.

4. During functional testing using Administrator credentials, ProDiscover IR did not recognize VSCs from a VHD format disk image file (i.e., it only allowed viewing of the VSCs that were mounted from a physical drive).

Overall, the *ProDiscover* methodology offered significant potential for quickly mounting all VSCs, performing GUI-enhanced analysis, extracting metadata from all VSCs, and exporting results. Some of the best of breed capabilities included in this utility provide great efficiencies and effectiveness for digital investigations.

This section described the most popular methods of achieving automation for VSC metadata/data extraction, analyzed the methods, and discussed both the benefits and limitations of each approach. Additional IV&V testing data is available in Appendix C. The following section provides analysis of common strengths and weaknesses of the

various approaches; it also provides an assessment of the enhancements which will solve the challenges identified in this research.

V. Merits and Limitations Analysis Confirms Requirements and Drives Enhancements

Merits and Limitations Analysis

Section III reviewed and provided analysis of several manual methods commonly used by digital investigators for accessing VSCs and extracting VSC metadata/data. Section IV provided information on several automated methods, thus showing evolution and efficiencies. IV&V testing of all approaches determined that most, if not all, approaches require additional enhancements in order to access VSCs and extract all VSC metadata/data in an automated fashion.

Analysis of the merits and limitations identified from all approaches provided a mechanism for comparison and contrast to confirm required capabilities, confirm shortcomings that must be eliminated, and identify must-have enhancements for developing a more robust solution. This review also provided insight into additional future areas of enhancement and expanded research. The current section will summarize the requirements presented for this research, then highlight three common challenges/limitations discovered in all current methodologies, and finally, discuss the benefits of multiple approaches, with the goal of developing best-of-breed capabilities and enhancement opportunities.

The requirements this research aimed to address include:

1. The VSC metadata/data extraction method must offer an automated approach for processing multiple (all) VSCs.

2. The user may process metadata/data for timeline generation and change visualization purposes (Phase 1). Next, the user may select arbitrary *investigator-targeted* file(s)/folder(s) for extraction and analysis (Phase 2, future work).

3. The data store resulting from metadata extraction must allow the user to select an arbitrary scope of data and view that data in any way the user desires (via extensible database queries against the dataset).

The research conducted in Sections III and IV identified common limitations that must be overcome in order to avoid falling short of satisfying the aforementioned requirements. They include the following:

1. All current methods require manual interaction outside of the tool/method employed in order to access the disk image containing all VSCs. Additionally, once accessed, limitations affecting the ability to automate the extraction of metadata/data from all VSCs on the disk image must be eliminated.

Enhancing the automation of disk mounting and the processing of all VSCs is a requirement for overcoming the current limitations. The method employed for VSC metadata/data extraction should offer an innate ability to access disk images and then cannot be limited to singular or iterative processing of VSCs; rather, it must be able to extract metadata/data from all VSCs on a mounted drive or disk image in an automated fashion.

2. Several methods extract only data files or limited aspects of metadata in support of timeline analysis. This presents several disadvantages: By extracting only data files or by extracting data files prior to extracting metadata, a digital investigator must invest both the storage requirements and time to recover this

data from the VSCs. As an example, even with a small 100GB volume as the dataset, if complete data extraction was required prior to metadata extraction, a digital investigator could be forced to wait while 1 to *n* VSCs worth of restored data were extracted. This may require significantly more storage than if the method allowed the digital investigator to initially extract only metadata, allowing subsequent focus on the data files and folder structure if/when warranted.

Another key limitation is with methods that only extract limited aspects of metadata; these methods must be enhanced to present metadata that is rich in the areas sought after by digital investigators. The resulting dataset must contain ample directory-tree structure, directory-tree content, and file attribute information, allowing for timeline analysis to compare/contrast both change and lack of change.

3. Most current methods extract metadata to some form of text-based log that is not inherently conducive to extensible analysis and comparison of the results. In order to provide digital investigators with the flexibility required to conduct thorough timeline analysis, this area requires enhancement to extract the resulting dataset to a database that allows extensible queries to be performed for any requirements desired. For example, if metadata is extracted to a database storage format, digital investigators would have the ability to visualize the metadata and execute queries to extract any subset of metadata they deemed valuable. Whether the analysis requires sorting the data by time characteristics, file attributes, or another method, having the extracted metadata in a storage format allowing these types of analyses is critical.

In addition to satisfying requirements and overcoming the common limitations, the method employed should offer the type of extensibility and flexibility that is commensurate with use in current and future digital investigations, by including the following enhancements:

1. the flexibility to execute any arbitrary third party utility,
2. the flexibility of an open-source, high-level programming language, and
3. platform flexibility (e.g., such as flexible execution from both a Microsoft Windows Operating System environment and potentially from Incident Responders' environments).

Of all the approaches and utilities examined, ProDiscover was deemed to be most advantageous from an ease-of-use perspective and for offering comprehensive analytical capabilities. ProDiscover showcased an impressive ability to mount individual VSCs and then view multiple VSCs simultaneously. It then visually depicted all VSC contents, requiring only the initial point-and-click user input. It also demonstrated strong capabilities for comparing VSC contents based on either hash or timestamp evaluation measures. Unfortunately, ProDiscover requires iterative point-and-click direction in order to process all VSCs of a particular image/drive. It also presented a challenge when attempting to mount the VSCs of a VHD format disk image file – it did not mount the VSCs during IV&V testing. Finally, its code base is closed-source (proprietary) and was only available via a demo license or at a cost of approximately $8,000.00 US dollars per license.

Of all open-source methodologies, *shadowcopy.py* exhibited the most merit. It performed most of the heavy lifting desired to solve this research's problem statement with only

moderate improvements needed to address shortcomings leading to the three limitations identified. With regard to desired functionality, *shadowcopy.py* offered the inherent flexibility of accessing and extracting data from multiple VSCs in sequential order, yet in an automated fashion. Its Python code offers programmers an open-source, high-level programming language with the flexibility to execute any arbitrary third party utilities at any time and with only slight code modifications. *Shadowcopy.py* supports digital investigations with its deduplication [42] of extracted data using customizable filename delimiters as well as with its ability to identify and process all non-local VSCs. *Shadowcopy.py* also utilized the ShadowVolume2.py code, which employed *mklink* and *vssadmin*, therefore already making use of these previously evaluated methods for accessing VSCs. Finally, as a possible extension, the potential use of Portable Python [43], which is preconfigured to allow Python code to be executed in a Windows environment from any USB storage device, along with *shadowcopy.py*, could provide even more future flexibility and rationale for using *shadowcopy.py* on removable media and Incident Responder's toolkits.

With regard to moderate improvements needed for addressing the problem statement, proposed *shadowcopy.py* enhancements include: automating the disk image access/mounting process, enhancing *shadowcopy.py*'s metadata extraction process, and improving *shadowcopy.py*'s reporting mechanism. The flexibility of the Python language and its cross-platform support made *shadowcopy.py* an easy programmer's choice for supporting these enhancements. Finally, to ensure *shadowcopy.py* enhancements could address the complete list of limitations identified during the evaluation phase (Appendix D), a comparison was performed by cross-referencing proposed *shadowcopy.py*

enhancements against all limitations identified in the evaluation phase of this research. Several limitations were deemed not applicable to the existing *shadowcopy.py* methodology. Other evaluation shortcomings were recognized limitations; however, they could be overcome with proposed *shadowcopy.py* enhancements. Enhancing the *shadowcopy.py* capability to support this research's goal of performing automated metadata/data extraction from multiple VSCs provides the best chance of succeeding in solving the problem statement.

Based on the numerous merits presented by the existing *shadowcopy.py* methodology as well as the proposed enhancements' ability to address identified limitations, the *shadowcopy.py* methodology was selected as the final candidate approach for enhancement to solve the problem statement of this research. Exploring areas of enhancement via custom *shadowcopy.py* modifications is further discussed in the following section.

VI. Custom Modifications Extend Automation

Exploration of Advancements

The preceding sections summarized the approaches associated with accessing VSCs and performing VSC metadata/data extraction; the sections also described the limitations identified during analysis of the access and extraction approaches. The evolution of the methods and utilities that are actively used by digital investigators was discussed and then insight was provided into potential areas of improvement. In order to advance VSC analysis, especially in support of digital investigations, this section will explore three areas of improvement. It will provide concrete examples of methodology and then will present solutions that address the limitations identified in Sections III-V. This section will also briefly touch upon the merits of additional *shadowcopy.py* enhancements which would further expand the scope of work toward resolving the problem statement.

Utilities Used

Table 4 highlights the utilities/methods used in support of the exploration of the three advancement areas.

Utilities/Methods	Description	Notes
Microsoft Windows 7 Professional, Service Pack 0 and Service Pack 1	Operating System	64-bit and 32-bit variants were used for testing purposes
VMWare Workstation, version 7.1.6-744570	Hypervisor (virtual machine manager)	www.vmware.com
Python interpreter, version 3.2	High-level programming language interpreter	www.python.org
PSTools version 2.44	Suite of tools for managing local and remote systems	PSexec.exe obtained NT Authority\System privileges
Shadowcopy.py	Original *shadowcopy.py* script	Used for testing and enhancements
Microsoft DiskPart version 6.1.7601	Virtual disk mounting utility	Standard with Windows 7 Professional OS
Microsoft *attrib.exe*	Utility for displaying file	C:\Windows\System32\attri

	and folder attributes	b.exe
Other utilities/methods	All other utilities/methods noted in Sections III-V	Required for IV&V testing purposes

Table 4: Utilities/methods supporting the exploration of advancements

The testing environment consisted of both the 64-bit and 32-bit variants of the Windows 7 Operating System, as well as the VMWare Workstation platform. This research effort also relied on the Python interpreter, Microsoft's *psexec.exe* utility, Microsoft's *diskpart* utility, Microsoft's *attrib.exe* utility, and the *shadowcopy.py* code. Having provided a high-level description of the utilities/methods used to create the environment for the exploration of advancements, this research effort will now provide a detailed methodology as well as lessons learned for exploring solutions to the following three problems:

1. automating disk image access/mounting (automating *shadowcopy.py*'s disk image access and mounting),
2. enhancing automated metadata extraction (enriching *shadowcopy.py*'s metadata and enhancing *shadowcopy.py*'s automated metadata extraction mechanism), and
3. exporting extracted metadata into a storage format that offers extensible queries and comparison of metadata from all VSCs (storing *shadowcopy.py*'s report in a format conducive to extensible queries and flexible metadata analysis).

Automating Disk Image Mounting

A challenge identified by many of the aforementioned methodologies and then manually implemented by several was the need for mounting VHD format or raw (*dd*) format disk image files using the Microsoft Windows 7 Computer Management Interface (including the corresponding Disk Manager element) or Microsoft's *diskpart* utility. Identifying an automated method of enhancing the manual implementation is an extension of previous

work and was listed as the first item of "future work" in Hom's ShadowCopy report.[35] Refining this technique into an automated disk image mounting mechanism for *shadowcopy.py* was critical in order to satisfy the first element of automating VSC metadata/data extraction; in response, the following advancements were made to *shadowcopy.py* using the *diskpart* utility methodology:

1. Partially-automated mounting and un-mounting, using *diskpart* scripts.

Testing was performed to identify manual methods of implementing the *diskpart* utility to successfully mount and un-mount a VHD format disk image file. Initial testing of *diskpart* revealed the *diskpart* command's *attach* argument could mount a VHD file. In order to do so, the disk image must first be the focus of the *diskpart* utility, or selected. The *select* argument was used to focus the *diskpart* utility on an arbitrary VHD file. Combining the steps in the proper order produced the following syntax:

SELECT VDISK FILE=<Volume>:\<Disk image filename.vhd>

ATTACH VDISK READONLY

The *readonly* argument allowed mounting of the VHD file, without altering its state, which performed the same function as selecting the Read-only checkbox in the Computer Management interface methodology (depicted inside the red box in Figure 19).

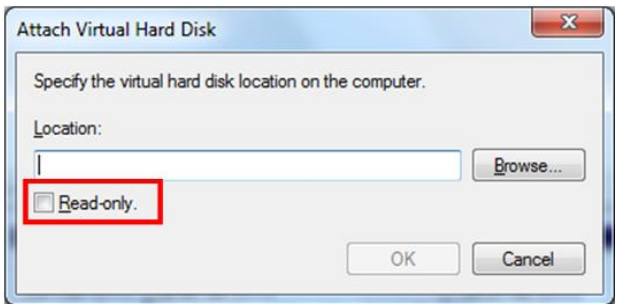


Figure 19: Computer Management interface's "Attach Virtual Hard Disk" element

Executing the script from the command line determined it successfully selected and attached the VHD file, as depicted in Figure 20.

```
D:\>diskpart /s "d:\ShadowCopy Testing and Results\diskpart_script.txt"
Microsoft DiskPart version 6.1.7601
Copyright (C) 1999-2008 Microsoft Corporation.
On computer: THESISVM-PC
DiskPart successfully selected the virtual disk file.
  100 percent completed
DiskPart successfully attached the virtual disk file.
```

Figure 20: Execution of a *diskpart* script to select and attach a virtual disk image (VHD) file

Next, the methodology and a script were developed for un-mounting the VHD file. The methodology utilizes the *select* and *detach* arguments, as noted below:

SELECT VDISK FILE=<Volume>:\<Disk image filename.vhd>

DETACH VDISK

Executing the script from the command line determined it successfully selected and detached the VHD file, as depicted in Figure 21.

```
D:\>diskpart /s "d:\ShadowCopy Testing and Results\unmount_diskpart_script.txt"
Microsoft DiskPart version 6.1.7601
Copyright (C) 1999-2008 Microsoft Corporation.
On computer: THESISVM-PC
DiskPart successfully selected the virtual disk file.
DiskPart successfully detached the virtual disk file.
```

Figure 21: Execution of a *diskpart* script to select and detach a virtual disk image (VHD) file

2. Integrating a mounting and un-mounting methodology into *shadowcopy.py*.

The *diskpart* script approach was incorporated into the *shadowcopy.py* script to implement disk image mounting automation that would support *shadowcopy.py*'s existing VSC access/mounting automation. To effectively integrate the new methodology, the *shadowcopy.py* Python code was modified with *–mount* and *–unmount* options. When executed with the *–mount* and *–unmount* options, the *shadowcopy.py* enhancement prompts the digital investigator for the path to the VHD file, verifies the path exists, and then calls *diskpart* to mount/unmount it, as depicted in Figure 22, below. (Note: The *shadowcopy.py* utility previously included an *–image* option. The *–image* option was removed and the image verification process was incorporated into the *–mount* code via the *image()* function.)

```
D:\>C:\Python32\python.exe shadowcopy.py --mount
Please enter the path of the vdisk (VHD) file to mount. -->d:\testimage.vhd
You entered: d:\testimage.vhd
Is this correct? (Enter 'Y' or 'y') -->y
The vdisk (VHD) path has been saved as: d:\testimage.vhd
…
DiskPart successfully attached the virtual disk file.

Please enter next command:
Usage: usage: shadowcopy.py [options] <EXTRACT-DIR>
<imagefile> may be a .vhd or a .raw.    If it is a .raw, it will
be converted to a .vhd IN PLACE, so be sure you have enough disk and the vhdtool.exe to do the
conversion
Note: this script must be run as administrator.
Options:
  -h, --help          show this help message and exit
  --mount             Prompts the user for a vdisk (VHD) or raw (DD) image
                      (converts to VHD format if necessary).  Then, mounts
                      the selected image.
  --list              Show the shadow volumes that are available.
  --local             Analyze only the local machine
  --maxsize=MAXSIZE   Specifies maximum size of a file to extract
  --minsize=MINSIZE   Specifies minimum size of a file to extract
  --noextract         Do not extract the shadow data
  --reportfn=REPORTFN Specify report output filename
  --zap               Overwrite report file if it exists
  --unmount           Unmount a selected VHD image
```

Figure 22: Execution of shadowcopy.py to select and attach a virtual disk image (VHD) file

The *shadowcopy.py –mount* and *–unmount* method was automated to write the user-entered, *image*-validated path to two *temporary* diskpart script files. The path is stored in the diskpart script files during *shadowcopy.py* execution and is then used for the –unmount process. After the –unmount process completes, the enhanced *shadowcopy.py* script removes both temporary diskpart script files from the analysis system. This enhancement, allowing *shadowcopy.py* to mount a VHD file in an automated manner, combined with its inherent capability of automatically processing all VSCs, satisfied the automated disk image access/mounting requirement.

Enhancing Automated Metadata Extraction

A second area for improvement was the ability to capture/extract all common directory structure, file, timestamp, and attribute metadata, which are commonly used by digital investigators when conducting timeline analysis or when determining whether certain system artifacts warrant further review during a digital investigation. This issue was identified as a limitation to many approaches of extracting VSC metadata, including the *shadowcopy.py* approach.

As previously implemented, the *shadowcopy.py* methodology did not incorporate directory structure, timestamp, or attribute information--which could be critical for a digital investigator--into the report file. However, of all the open-source utilities tested, *shadowcopy.py* showcased the best inherent approach for automating a combined metadata/data extraction capability. Consequently, the *shadowcopy.py* metadata/data extraction methodology (reporting feature) was selected as a noteworthy candidate for additional enhancement.

In an effort to integrate increased functionality into the existing *shadowcopy.py* reporting mechanism, the enhancement approach focused on incorporating directory structure, timestamp, and attribute metadata, which are commonly deemed helpful in support of timeline analysis and digital investigations. Portions of the *shadowcopy.py* code were modified to capture and report the additional required metadata. An initial modification of the *shadowcopy.py* script tested execution of *LogParser* within the *shadowcopy.py* environment; however, the approach required the additional step of remounting VSCs using the *mklink* methodology. In order to achieve the same functionality without adding additional steps, the *shadowcopy.py* script was modified to incorporate the required directory structure, timestamp, and attribute enhancements using Python code and a native Microsoft Windows 7 executable, *attrib.exe*. The enhancements are as follows:

1. To record directory structure information into the report, the enhancements to *shadowcopy.py* consisted of re-aligning code for efficiency and then writing new directory structure processing functionality into the *process()* function within the *shadowcopy.py* script. Similar to the *shadowcopy.py* Python code that processes filenames, the directory structure processing code performs the following functions:

    a. executes the *stat* command, which is used to gather information used to record the mtime, atime, ctime, size, etc, for each directory,

    b. records the attributes of each directory,

    c. records the originating system name and VSC name for each directory,

    d. records MAC times for each directory, and

e. records errors, such as "Access is denied" and "The media is write protected," as it processes each directory.

The new functionality records the path and directory name metadata into the report, in addition to previously recorded filename metadata. The value this enhancement creates is the most complete picture by combining a record of the system's folder structure with the existing record of the system's files. The enhanced structure, combined with timestamps and attributes, will better help digital investigators depict a system's historical changes. The red box around the second column, "Path," in Figure 23, depicts directory paths from the enhanced report. The blue and green boxes in the seventh column, "MTime," coupled with the second red box in the sixth column, "Volume," depict how the "\Users\VSCTest\AppData\Local\Microsoft\Windows" directory has changed between VSCs 3 and 4. Additionally, the different MD5 hashes for the UsrClass.dat file show its contents changed across all four VSCs and the MTime changes provide the timestamp of the data changes.

id	Path	MD5	Size	Machine	Volume	M Time
2858	\Users\VSCTest\AppData\Local\Microsoft\Windows	0 (dirpath)	4096	VSCTest-PC	HarddiskVolumeShadowCopy2	2011-11-06 20:10:25.197750
2859	\Users\VSCTest\AppData\Local\Microsoft\Windows\UsrClass.dat	56dfc9a4d2849ded54a7944c03d1346a	262144	VSCTest-PC	HarddiskVolumeShadowCopy2	2011-11-06 20:08:58.635250
81208	\Users\VSCTest\AppData\Local\Microsoft\Windows	0 (dirpath)	4096	VSCTest-PC	HarddiskVolumeShadowCopy3	2011-11-06 20:10:25.197750
81209	\Users\VSCTest\AppData\Local\Microsoft\Windows\UsrClass.dat	457f895b5127fa4df14ca0ba47013c0a	262144	VSCTest-PC	HarddiskVolumeShadowCopy3	2011-11-06 20:21:41.229000
159662	\Users\VSCTest\AppData\Local\Microsoft\Windows	0 (dirpath)	4096	VSCTest-PC	HarddiskVolumeShadowCopy4	2011-11-06 21:04:32.357422
159663	\Users\VSCTest\AppData\Local\Microsoft\Windows\UsrClass.dat	12dfacc8028ccc323c36c9f214e885a5	262144	VSCTest-PC	HarddiskVolumeShadowCopy4	2011-11-06 21:14:39.932617
248282	\Users\VSCTest\AppData\Local\Microsoft\Windows	0 (dirpath)	4096	VSCTest-PC	HarddiskVolumeShadowCopy5	2011-11-06 21:04:32.357422
248283	\Users\VSCTest\AppData\Local\Microsoft\Windows\UsrClass.dat	95fc6838ab51fb9ec68a32e5ff6b472e	262144	VSCTest-PC	HarddiskVolumeShadowCopy5	2011-11-06 21:14:39.932617

Figure 23: Enhanced shadowcopy.py report capturing directory structure information.

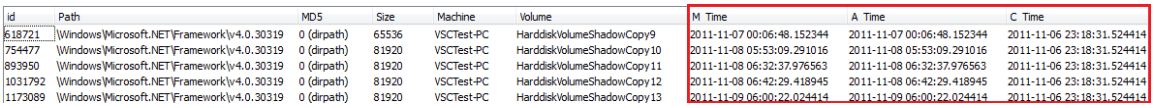
2. To incorporate timestamp information into the report, the enhancements to *shadowcopy.py* code were minimal. Incorporating the following three lines of Python code into the *process()* function within the *shadowcopy.py* script introduced MAC time metadata into the report.

datetime.datetime.fromtimestamp(st.st_mtime),

datetime.datetime.fromtimestamp(st.st_atime),

datetime.datetime.fromtimestamp(st.st_ctime),

Note: The Python class datetime.datetime provides a combination of date and time output whereas datetime.date and datetime.time provide only the respective individual date or time elements. Figure 24 depicts the MTime, ATime, and CTime fields in the enhanced report.

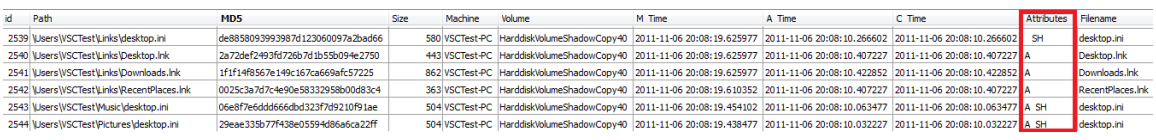


id	Path	MD5	Size	Machine	Volume	M Time	A Time	C Time
618721	\Windows\Microsoft.NET\Framework\v4.0.30319	0 (dirpath)	65536	VSCTest-PC	HarddiskVolumeShadowCopy9	2011-11-07 00:06:48.152344	2011-11-07 00:06:48.152344	2011-11-06 23:18:31.524414
754477	\Windows\Microsoft.NET\Framework\v4.0.30319	0 (dirpath)	81920	VSCTest-PC	HarddiskVolumeShadowCopy10	2011-11-08 05:53:09.291016	2011-11-08 05:53:09.291016	2011-11-06 23:18:31.524414
893950	\Windows\Microsoft.NET\Framework\v4.0.30319	0 (dirpath)	81920	VSCTest-PC	HarddiskVolumeShadowCopy11	2011-11-08 06:32:37.976563	2011-11-08 06:32:37.976563	2011-11-06 23:18:31.524414
1031792	\Windows\Microsoft.NET\Framework\v4.0.30319	0 (dirpath)	81920	VSCTest-PC	HarddiskVolumeShadowCopy12	2011-11-08 06:42:29.418945	2011-11-08 06:42:29.418945	2011-11-06 23:18:31.524414
1173089	\Windows\Microsoft.NET\Framework\v4.0.30319	0 (dirpath)	81920	VSCTest-PC	HarddiskVolumeShadowCopy13	2011-11-09 06:00:22.024414	2011-11-09 06:00:22.024414	2011-11-06 23:18:31.524414

Figure 24: Enhanced *shadowcopy.py* report showing MTime, ATime, and CTime fields.

This example depicts change to the MTime and ATime, relative to all five VSCs (HarddiskVolumeShadowCopy9 - 13), providing an example of historical artifacts that are now available to digital investigators through analysis of multiple VSCs. This capability and the findings it produces will enhance timeline generation and historical analysis methods.

3. To incorporate attribute information into the report, the enhancements to *shadowcopy.py* consisted of adding Python code and a call for the execution of *attrib.exe* from the *process()* function within the *shadowcopy.py* script. The new functionality records the attributes of files and folders. Figure 25 depicts the enhanced report, with the attribute information residing in the seventh column, "Attributes."



id	Path	MD5	Size	Machine	Volume	M Time	A Time	C Time	Attributes	Filename
2539	\Users\VSCTest\Links\desktop.ini	de885809399398 7d123060097a2bad66	580	VSCTest-PC	HarddiskVolumeShadowCopy40	2011-11-06 20:08:19.625977	2011-11-06 20:08:10.266602	2011-11-06 20:08:10.266602	SH	desktop.ini
2540	\Users\VSCTest\Links\Desktop.lnk	2a72def2493fd726b7d1b55b094e2750	443	VSCTest-PC	HarddiskVolumeShadowCopy40	2011-11-06 20:08:19.625977	2011-11-06 20:08:10.407227	2011-11-06 20:08:10.407227	A	Desktop.lnk
2541	\Users\VSCTest\Links\Downloads.lnk	1f1f14f8567e149c167ca669afc57225	862	VSCTest-PC	HarddiskVolumeShadowCopy40	2011-11-06 20:08:19.625977	2011-11-06 20:08:10.422852	2011-11-06 20:08:10.422852	A	Downloads.lnk
2542	\Users\VSCTest\Links\RecentPlaces.lnk	0025c3a7d7c4e90e58332958b00d83c4	363	VSCTest-PC	HarddiskVolumeShadowCopy40	2011-11-06 20:08:19.610352	2011-11-06 20:08:10.407227	2011-11-06 20:08:10.407227	A	RecentPlaces.lnk
2543	\Users\VSCTest\Music\desktop.ini	06e8f7e6ddd666dbd323f7d9210f91ae	504	VSCTest-PC	HarddiskVolumeShadowCopy40	2011-11-06 20:08:19.454102	2011-11-06 20:08:10.063477	2011-11-06 20:08:10.063477	A SH	desktop.ini
2544	\Users\VSCTest\Pictures\desktop.ini	29eae335b77f438e05594d86a6ca22ff	504	VSCTest-PC	HarddiskVolumeShadowCopy40	2011-11-06 20:08:19.438477	2011-11-06 20:08:10.032227	2011-11-06 20:08:10.032227	A SH	desktop.ini

Figure 25: Enhanced shadowcopy.py report showing attributes field.

Two additional enhancements were made to improve metadata extraction and provide additional efficiency. As previously engineered, *shadowcopy.py* reported metadata for files against which it could execute the *stat* command. *Shadowcopy.py* processed errors as it attempted to stat every file, some of which were caused by permissions or access issues. In order to enhance output, *shadowcopy.py* testing was executed using *NT Authority\System* permissions. To obtain *NT Authority\System* permissions, testing consisted of issuing the *psexec -hsi cmd* command and then executing the *shadowcopy.py* script from within the new command window. When *shadowcopy.py* was executed as *NT Authority\System* during testing, the number of output records from a 40GB source dataset increased by approximately 0.03%, from 96,525 records to 96,553 records for an arbitrary test. This does not represent a significant increase in production; however, in the world of digital investigations, it may prove to be the differentiator in identifying inculpatory or exculpatory evidence.

Finally, minor edits were made within the *process* function of the *shadowcopy.py* code to streamline the operation of the code that calls the *make_filename_distinct* function. For example, if the *noextract* option was selected, files would not be extracted from VSCs; therefore, the need to determine a destination directory for those files or to create a distinct filename may be eliminated. To allow for more efficient execution, minor segments of this code were removed and reinserted after *shadowcopy.py*'s code checked for the absence of the *noextract* option – at the point where execution of code that checks for a destination directory and distinct filenames is more prudent.

Storage Format/Method

As discussed in preceding sections, several VSC metadata/data retrieval methodologies recover VSC metadata to a custom report or other common forms of delimiter-separated values format, such as a tab-separated values format or comma-separated values format. In order to support automated export and storage of metadata information from multiple VSCs as well as provide a mechanism that offers adaptable data selection criteria and extensible queries, research into another format/method of storage was required. Specifically, it would be most beneficial to follow-on research efforts if the *shadowcopy.py* reporting mechanism output data were in a format that allowed any imaginable type of query to be issued, and the customization of those queries allowed for the production of one-to-many records of output (whichever is desired/needed). This requirement may be met by storing the data in a SQL format such as that of a SQL server implementation or SQLite.

Metadata Storage in Database Format (SQLite)

To support the goal of exporting all VSC metadata into SQL storage, review of the various utilities determined the *LogParser* utility innately supported exporting results to SQL format. Additionally, the Python interpreter supports collaboration with the SQLite database format; thus, testing and implementation was conducted to determine whether the *shadowcopy.py* method could export metadata to SQLite format."[44] Additionally, research was conducted into supporting a combined *shadowcopy.py*/*LogParser* method that could export metadata to SQLite format.

The Python interpreter, version 2.5 and newer, natively includes SQLite support via the "import sqlite3" command. Based on the relative ease of incorporating this native

functionality into the *shadowcopy.py* methodology, it was selected as the desired method for enhancing the reporting function. Additionally, since SQLite storage format allows anyone reviewing the data, especially digital investigators conducting timeline analysis or visualizing change, to issue any type of SQL query imaginable to produce one-to-many records of output, it became an ideal candidate.

To support storing VSC metadata in SQLite format, the SQLite connect API was used to create/open a database file and provide its file handle. The *shadowcopy.py* code was modified to change the *reportfn* variable to assign the new database filename. A cursor was assigned to allow the *shadowcopy.py* code to iterate through the database contents. Next, three functions, *createTable(), addRecord(),* and *deleteTable()*, were added to the *shadowcopy.py* code. Functions *createTable()* and *deleteTable()* are used to create and delete, respectively, a new table named ShadowCopy in the database file specified by the connect API. The ShadowCopy table contains an "id" record number variable of integer type that is used as the primary key, as well as the following variables of text or integer type: path, MD5, size, machine, volume, m_time, a_time, c_time, attributes, and filename. The *addRecord()* function is called from the existing *process()* function within *shadowcopy.py* to report directory, file, time, and attribute metadata as the *process()* function "walks" through each VSC. After each VSC is processed, the changes are committed to the database and, when all VSCs are processed, the cursor to the database file is closed. In Figure 26, a SQLite browser depicts sample output from *shadowcopy.py*'s report (SQLite database) containing extracted VSC metadata, which has been queried for filenames that match "index.dat."

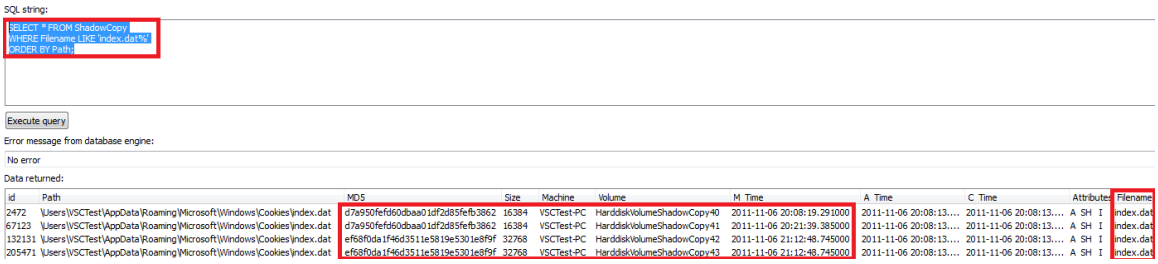
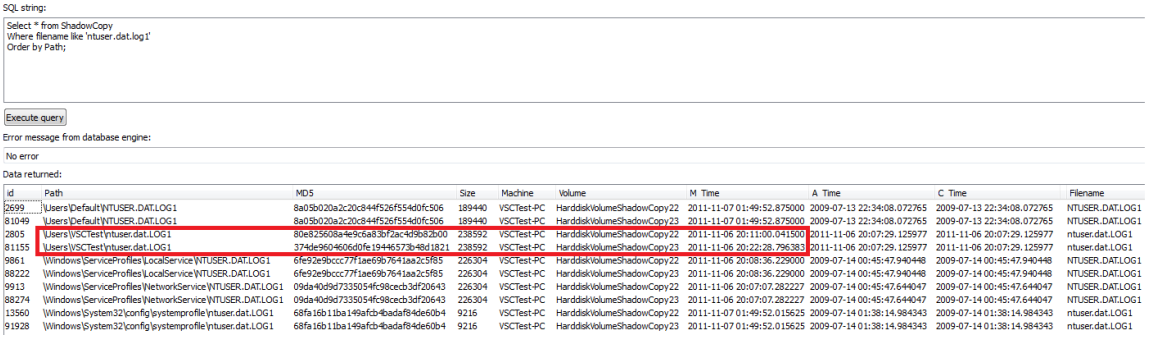


Figure 26: SQLite metadata output from an arbitrary VSC file

This example shows the *shadowcopy.py* output (report) is now enhanced to record metadata in SQLite format, which allows anyone reviewing the data, especially digital investigators conducting timeline analysis or visualizing change, to issue any type of SQL query imaginable to produce one-to-many records of output. The sample query executed to produce the above result was "SELECT * FROM ShadowCopy WHERE Filename LIKE 'index.dat%' ORDER BY Path." Review of the resulting metadata identifies changes to the index.dat file's MD5 hash and MTime across VSCs HarddiskVolumeShadowCopy40 – 43. This provides an example of how, using an open source utility to show historical changes to a system, digital investigators may depict multiple versions of metadata artifacts, after extracting them from multiple VSCs. The arbitrary result above is indicative of the data that will support timeline generation and visualization of change, both of which are possible using metadata extracted from multiple VSCs.

After reviewing the enhanced metadata output and identifying files/folders of interest that require more in-depth analysis, a digital investigator would link the metadata results extracted from *shadowcopy.py*'s output with the extracted data results. In order to accomplish this, a digital investigator may use the MD5 hash value to make that critical link. For example, in Figure 27, extracted metadata results show the "ntuser.dat.log1"

file changed between HarddiskVolumeShadowCopy22 and 23; therefore, it was identified as an item of interest.



SQL string:

```
Select * from ShadowCopy
Where filename like 'ntuser.dat.log1'
Order by Path;
```

Execute query

Error message from database engine:

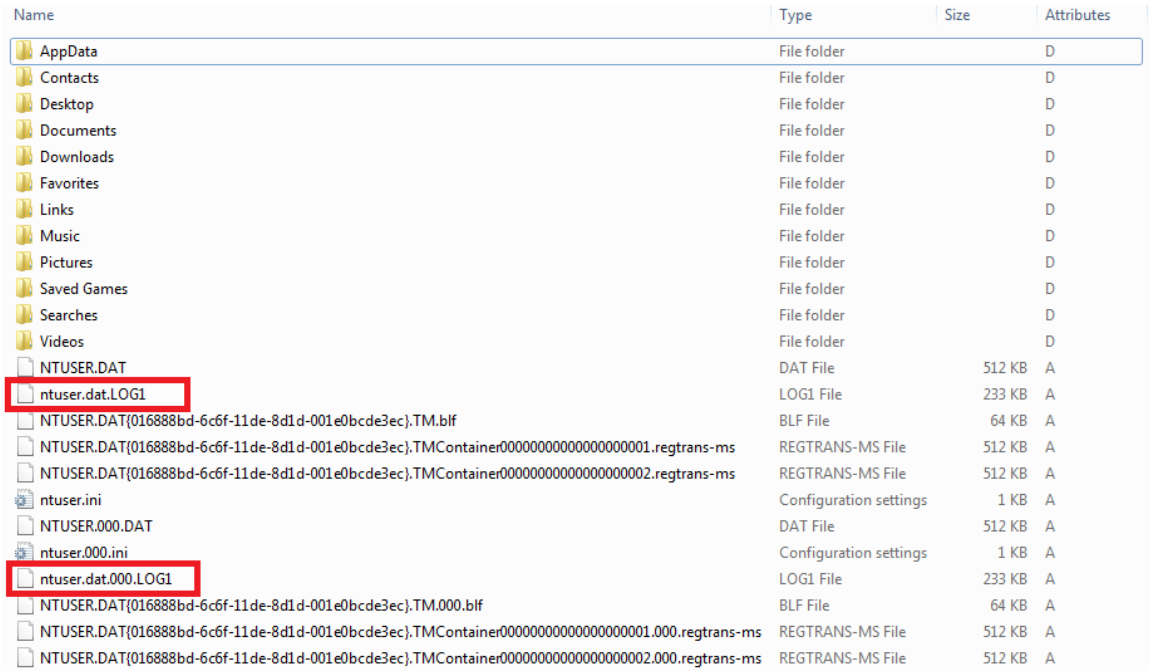
No error

Data returned:

id	Path	MD5	Size	Machine	Volume	M Time	A Time	C Time	Filename
2699	\Users\Default\NTUSER.DAT.LOG1	8a05b020a2c20c844f526f554d0fc506	189440	VSCTest-PC	HarddiskVolumeShadowCopy22	2011-11-07 01:49:52.875000	2009-07-13 22:34:08.072765	2009-07-13 22:34:08.072765	NTUSER.DAT.LOG1
81049	\Users\Default\NTUSER.DAT.LOG1	8a05b020a2c20c844f526f554d0fc506	189440	VSCTest-PC	HarddiskVolumeShadowCopy23	2011-11-07 01:49:52.875000	2009-07-13 22:34:08.072765	2009-07-13 22:34:08.072765	NTUSER.DAT.LOG1
2805	\Users\VSCTest\ntuser.dat.LOG1	80e825608a4e9c6a83bf2ac4d9b82b00	238592	VSCTest-PC	HarddiskVolumeShadowCopy22	2011-11-06 20:11:00.041500	2011-11-06 20:07:29.125977	2011-11-06 20:07:29.125977	ntuser.dat.LOG1
81155	\Users\VSCTest\ntuser.dat.LOG1	374de9604606d0fe19446573b48d1821	238592	VSCTest-PC	HarddiskVolumeShadowCopy23	2011-11-06 20:22:28.796383	2011-11-06 20:07:29.125977	2011-11-06 20:07:29.125977	ntuser.dat.LOG1
9861	\Windows\ServiceProfiles\LocalService\NTUSER.DAT.LOG1	6fe92e9bccc77f1ae69b7641aa2c5f85	226304	VSCTest-PC	HarddiskVolumeShadowCopy22	2011-11-06 20:08:36.229000	2009-07-14 00:45:47.940448	2009-07-14 00:45:47.940448	NTUSER.DAT.LOG1
88222	\Windows\ServiceProfiles\LocalService\NTUSER.DAT.LOG1	6fe92e9bccc77f1ae69b7641aa2c5f85	226304	VSCTest-PC	HarddiskVolumeShadowCopy23	2011-11-06 20:08:36.229000	2009-07-14 00:45:47.940448	2009-07-14 00:45:47.940448	NTUSER.DAT.LOG1
9913	\Windows\ServiceProfiles\NetworkService\NTUSER.DAT.LOG1	09da40d9d7335054fc98cecb3df20643	226304	VSCTest-PC	HarddiskVolumeShadowCopy22	2011-11-06 20:07:07.282227	2009-07-14 00:45:47.644047	2009-07-14 00:45:47.644047	NTUSER.DAT.LOG1
88274	\Windows\ServiceProfiles\NetworkService\NTUSER.DAT.LOG1	09da40d9d7335054fc98cecb3df20643	226304	VSCTest-PC	HarddiskVolumeShadowCopy23	2011-11-06 20:07:07.282227	2009-07-14 00:45:47.644047	2009-07-14 00:45:47.644047	NTUSER.DAT.LOG1
13560	\Windows\System32\config\systemprofile\ntuser.dat.LOG1	68fa16b11ba149afcb4badaf84de60b4	9216	VSCTest-PC	HarddiskVolumeShadowCopy22	2011-11-07 01:49:52.015625	2009-07-14 01:38:14.984343	2009-07-14 01:38:14.984343	ntuser.dat.LOG1
91928	\Windows\System32\config\systemprofile\ntuser.dat.LOG1	68fa16b11ba149afcb4badaf84de60b4	9216	VSCTest-PC	HarddiskVolumeShadowCopy23	2011-11-07 01:49:52.015625	2009-07-14 01:38:14.984343	2009-07-14 01:38:14.984343	ntuser.dat.LOG1

Figure 27: Results showing changes to the "ntuser.dat.log1" file

A cursory review of the data extracted by *shadowcopy.py* shows two versions of the "ntuser.dat.log1" file extracted as "ntuser.dat.log1" and "ntuser.dat.000.log1," as depicted in Figure 28.



Name	Type	Size	Attributes
AppData	File folder		D
Contacts	File folder		D
Desktop	File folder		D
Documents	File folder		D
Downloads	File folder		D
Favorites	File folder		D
Links	File folder		D
Music	File folder		D
Pictures	File folder		D
Saved Games	File folder		D
Searches	File folder		D
Videos	File folder		D
NTUSER.DAT	DAT File	512 KB	A
ntuser.dat.LOG1	LOG1 File	233 KB	A
NTUSER.DAT{016888bd-6c6f-11de-8d1d-001e0bcde3ec}.TM.blf	BLF File	64 KB	A
NTUSER.DAT{016888bd-6c6f-11de-8d1d-001e0bcde3ec}.TMContainer00000000000000000001.regtrans-ms	REGTRANS-MS File	512 KB	A
NTUSER.DAT{016888bd-6c6f-11de-8d1d-001e0bcde3ec}.TMContainer00000000000000000002.regtrans-ms	REGTRANS-MS File	512 KB	A
ntuser.ini	Configuration settings	1 KB	A
NTUSER.000.DAT	DAT File	512 KB	A
ntuser.000.ini	Configuration settings	1 KB	A
ntuser.dat.000.LOG1	LOG1 File	233 KB	A
NTUSER.DAT{016888bd-6c6f-11de-8d1d-001e0bcde3ec}.TM.000.blf	BLF File	64 KB	A
NTUSER.DAT{016888bd-6c6f-11de-8d1d-001e0bcde3ec}.TMContainer00000000000000000001.000.regtrans-ms	REGTRANS-MS File	512 KB	A
NTUSER.DAT{016888bd-6c6f-11de-8d1d-001e0bcde3ec}.TMContainer00000000000000000002.000.regtrans-ms	REGTRANS-MS File	512 KB	A

Figure 28: Results showing two versions of the extracted "ntuser.data.log1" file

In order to definitively link both versions of the files with the extracted metadata results, an MD5 hashing utility, such as *md5deep*, may be used to create an MD5 hash of each file, as depicted in Figure 29.

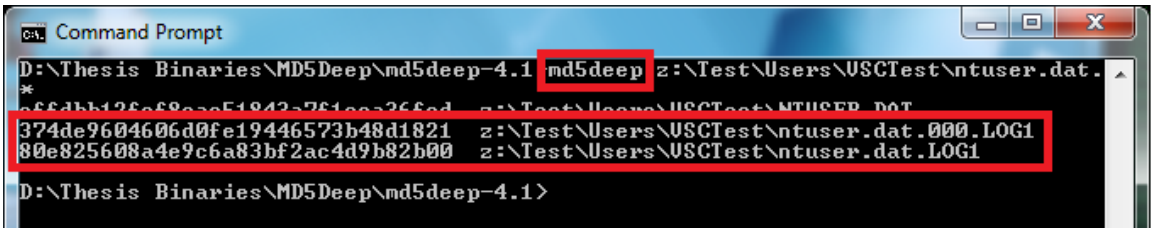


Figure 29: MD5 hashing utility definitively links extracted metadata to extracted data

Enhancement Results Summary

The enhancement case studies demonstrate this research successfully achieved the following improvements:

1. Automated disk image mounting: This improvement was achieved by modifying the *shadowcopy.py* script with a *diskpart* utility enhancement. During testing, this methodology operated in a consistent and repeatable manner for a simple VHD format disk image file, which, based on review of the *shadowcopy.py* script and Hom's ShadowCopy report, supported the utility's original intent.

2. Enhanced automated metadata extraction: The three reporting improvements listed below were achieved by enhancing the *shadowcopy.py* script.

    a. Reporting of directory structure information

    b. Reporting of MAC times information

    c. Reporting of file/folder attribute information

During testing, the added directory structure, MAC times, and attribute information was consistently produced in the *shadowcopy.py* report (database).

3. Enhanced reporting to a SQLite database format: This improvement was achieved by leveraging the Python interpreter's SQLite integration, allowing results to be stored in a SQLite flat-file format (<filename>.Db) that is capable of handling multiple extensible queries for the return of one-to-many data records. Results were viewed in a SQLite browser and were consistent with expectations.

Overall, the exploration of advancements, or case studies, produced results that showed increased richness of metadata as well as a significant decrease in storage requirements for metadata-only results, when compared with full data extraction. As an example, in order to obtain metadata to generate a timeline for an arbitrary 100GB dataset with 10 arbitrary VSCs, a digital investigator would not need to have 1TB of disk space available and the time required to extract all 10 versions of complete file and folder structure data. Rather, digital investigators can now anticipate the time required for extraction of multiple VSCs' metadata in terms of hours (even with limited processing power) and the use of multiple MBs to low GBs of storage space for the resulting report. Digital investigators can then view all output in the SQLite report and query any aspect of the data to obtain the desired view/output. Queries may be as complex or as simple as desired for supporting timeline generation or cataloging a system's historical artifacts. Combining these enhancements with the inherent *shadowcopy.py* capabilities offers:

a. the inherent flexibility of accessing and extracting metadata and data from multiple VSCs in an automated fashion,

b. open-source and high-level Python programming language that is flexible to execute any third party utilities,

c. an aid to digital investigators, with the deduplication of extracted data files and the ability to identify and process all non-local VSCs,

d. the inherent use of the *mklink* and *vssadmin* methods via the ShadowVolume2.py code,

e. enhanced VHD format disk image file mounting automation via *diskpart* enhancements, and

f. enhanced metadata processing and reporting mechanisms.

Although improvement in *shadowcopy.py*'s overall capability has been realized, the enhancement case studies demonstrate insight into the following issues, which currently remain unresolved:

1. Timeliness of the process: The original version of the *shadowcopy.py* script issued the *stat* command and calculated an MD5 hash for every file. During testing, with error reporting suppressed, the execution time against arbitrary VSCs averaged approximately 2.09 minutes/GB. After enhancements, the execution time against the same VSCs averaged approximately 3.43 minutes/GB. The increase in processing time is best attributed to the additional capture and reporting of all directory structure, timestamp, and attribute information into the SQLite report. Hence, decreasing the time required to process all VSCs from an arbitrary VHD format disk image file is a lingering issue that could benefit from additional research, including parallel processing techniques, as initially noted by Hom.[35]

2. Automated disk image mounting/unmounting functioned consistently in controlled test scenarios across several test systems; however, it is speculated that this mechanism would require additional adjustments to allow it to support complicated instances involving multiple VHD format disk image files. Testing multiple or complex source scenarios is an area that could benefit from additional research.

3. Further investigation into eliminating errors produced during *shadowcopy.py* processing, such as when the Windows Operating System cannot access a particular folder or file, is warranted. Other items that could produce errors and/or omissions also require further attention, such as:

a. *shadowcopy.py* does not currently process NTFS junctions or the files in a system's root directory. The root directory challenge is likely resolved via trivial code edits; however, the NTFS junction challenge may require additional research.

b. *attrib.exe* is not producing the "D" attribute indicative of directories. This challenge may also require further study.

c. Variables for reporting into the SQLite database are currently of integer and text type. The variable types should be reviewed and adjusted to eliminate overflows or other undesirable occurrences.

4. As written, *shadowcopy.py* is confined to work with the VSS API; however, in order to adopt the Metz, McKinnon, and ProDiscover method of parsing VSCs, *shadowcopy.py*'s metadata/data extraction component could be combined with

libvshadow or another method for a non-API-based metadata/data extraction solution.

5. As written, *shadowcopy.py* is designed to gather all metadata/data from all VSCs. (This was the boundary of the scope of the initial research concept.) *Shadowcopy.py* currently does not gather the metadata/data from the *live* volume in addition to the content from the VSCs; therefore, some may feel as though it does not capture all data potentially required for an investigation. The capability could easily be integrated by extracting data from *HarddiskVolume*[number] in addition to *HarddiskVolumeShadowCopy*[number].

6. Command-line interface execution: The *shadowcopy.py* script could benefit from a GUI overlay controlling, at the very minimum, the ability to browse paths for the selection of VHD format disk image files to process. A GUI could resolve additional issues, and is discussed further in Section VIII.

VII. Conclusion

Overview

The basis of this research consisted of an amalgamation of three areas, which are summarized as follows:

1. deficiencies in completeness were identified when generating timelines using only the current, or "available," version of artifacts from a computer system,

2. a backup mechanism of the Microsoft Windows 7 Operating System provides the ability to effectively preserve multiple versions of system artifacts; several methods already exist for accessing the backups, or VSCs, and extracting their data, thereby potentially mitigating the deficiencies, and

3. additional automation of the VSC access and metadata extraction methods was identified as an area needing focus, and thus was established as the area of concentration for this research.

First, this research noted a problem that existed during timeline generation when using only the available, or most recent, version of the artifacts from a computer system. The resulting timeline would likely provide a somewhat limited picture of historical changes for the system, especially when compared with a timeline generated using all versions of the artifacts from the same computer system. For example, if previous versions of artifacts and/or previous artifact metadata changes are overwritten and therefore not retained on a system, analysis of current artifacts, such as time/date stamps and operating system/program/registry artifacts, may provide only a limited picture of activities for the system.

Second, this research noted the Microsoft Windows Operating System's backup mechanism, which is capable of retaining multiple versions of data storage units for a system, effectively provides a highly-detailed record of system changes, which may be used as a data source to resolve the problem statement. As a potential solution, this research noted incorporating VSC metadata into a timeline as a potential aid to the problem statement, with the caveat that the data must be accessed and extracted in a consistent, repeatable, and if possible, automated manner. This research noted multiple methods exist for accessing VSCs and extracting metadata/data.

Third, this research aimed to identify the methodology, and subsequently, enhancements to automate accessing and extracting directory-tree and file attribute metadata from multiple VSCs of the Windows 7 Operating System. With VSC metadata extracted and recorded in a format that allowed extensible querying for output of one-to-many records, the data required to support enhanced timeline analysis could be made available in support of digital investigations.

Research Activities

Due to the limited amount of published materials available, this research first set out to provide the reader with a background and general understanding of VSS and VSCs. The goal of this portion of the research was to provide a limited, but structured review of several key facets of VSS and VSCs, which may benefit future research efforts. Next, this research presented an overview of several common methods for accessing VSCs and extracting metadata and data. Subsequently, the focus was sharpened toward highlighting existing methods of *automating* VSC metadata and data extraction. Next, IV&V of the common methods of VSC metadata/data extraction was performed, and then

the merits and limitations of each of the existing approaches were assessed. The *shadowcopy.py* script was selected as the preeminent candidate approach for enhancement to solve the problem statement of this research effort. Finally, improvements to the *shadowcopy.py* script were identified and implemented in order to advance the automation of accessing VSCs and performing VSC metadata extraction in support of timeline analysis:

1. automating disk image access/mounting (automating *shadowcopy.py*'s disk image access and mounting),
2. enhancing automated metadata extraction (enriching *shadowcopy.py*'s metadata and enhancing *shadowcopy.py*'s automated metadata extraction mechanism), and
3. exporting extracted metadata into a storage format that offers extensible queries and comparison of metadata from all VSCs (storing *shadowcopy.py*'s report in a format conducive to extensible queries and flexible metadata analysis).

Assessment of the Benefits to Timelines and the Visualization of Change

Section II establishes the significance of time/timelines in digital investigations and the ability of a machine's Basic Input/Output System (BIOS), operating system, and file system to function as the baseline for timekeeping and time-stamping of certain system artifacts. It also briefly describes how digital artifacts such as file attributes, directory-tree structure, directory-tree contents, and the conglomeration of those data may support current research in a related area, the visualization of change-over-time. This section culminates with an assessment of how the product of this research, enhancements to the automation of extracting VSC metadata, supports more comprehensive timeline generation as well as other projects, such as the visualization of change-over-time.

The tangible result of this research is the advancement of the methodology for automatically extracting VSC metadata and the storage of results in SQLite database format, allowing extensible queries for any subsequent need. The methodology supports timeline generation using a more comprehensive dataset, as the resulting timeline should be able to depict artifact metadata from all VSCs of a particular VHD format disk image file. This allows for the visualization of the progression of change, or lack thereof, for system artifacts across an arbitrary timeline. As the richness of the extracted metadata advanced to include directory structure, MAC times, and attribute information, the resulting dataset should add further depth and breadth to timelines created using VSC metadata/data. Another goal of this research is to provide enhancement value to other projects, such as those that visualize change-over-time, through the resultant dataset..

When considering the inherent file and folder structure export capability of the original *shadowcopy.py* script, coupled with the additional richness of the metadata that is now produced based on enhancing the script, it seems that together, the inherent and enhanced methodologies provide digital investigators with a more robust capability to export metadata from VSCs, analyze and/or depict items of interest, export complete data file/folder structure from VSCs, and analyze items of interest a final time. The combined methodology thus offers digital investigators the desired additional automation for processing VHD format disk image files and VSC data, as well as the ability to derive a more comprehensive chronological representation of a system's historical changes, by having the input of multiple instances of system artifacts.

VIII. Future and Related Work

This research effort pursued enhancing automated methods of accessing VSCs and extracting metadata/data from all VSCs on a particular disk image, all in support of timeline generation for digital investigations. The automation and reporting enhancements that were added to the *shadowcopy.py* script resulted in the ability to automatically mount disk images and then to extract directory structure, timestamp, and attribute-enhanced metadata into a SQLite database reporting format. Based upon the analysis conducted during this research, several areas to consider for future work include:

1. allowing interactive selection of *investigator-targeted* paths from which to extract data,

2. implementing *shadowcopy.py* as a stand-alone static binary with GUI support,

3. enhancing the interactive VHD format disk image file and VSC selection mechanism, and

4. exploring a cross-platform implementation.

Each of these areas is discussed in further detail below:

1. Phase 2 (Future Work), could improve upon the current *shadowcopy.py* approach by offering the investigator an interactive mechanism to custom-select one-to-many *investigator-targeted* paths to extract data from, based upon review of the initial metadata output. The enhancement would offer digital investigators the ability to sharpen their focus on items of interest more quickly by exporting and examining only items of interest from the digital evidence.

2. Incorporating the *shadowcopy.py* script into a stand-alone binary with an interactive GUI could be extremely beneficial for allowing the investigator to

browse to the path of the VHD format disk image file, as well as for review of extracted metadata. Along with the other Phase 2 enhancement described, this development could allow for digital investigators to review metadata results in a GUI format and then use the GUI to select the *investigator-targeted* data to extract for subsequent analysis.

3. Providing interactive advancements for VHD format disk image file selection and VSC selection could be an extremely beneficial future development. This would be useful in the event a digital investigator should need to conduct parallel analysis of multiple VHD format disk image files that each contain multiple VSCs. For example, combining the portions of the *shadowcopy.py* script that extract metadata/data with portions of the libvshadow project, which mounts VSCs via open source methods on the Linux platform, may be an extremely beneficial approach for mounting any combination of VHD format disk image files and VSCs.

4. An automated framework implementation that allows digital investigators to mount/access VSCs from a variety of operating system platforms, such as Linux, OS X, etc, could be a remarkably useful future enhancement. The original version of *shadowcopy.py* claimed it was portable such that it could be employed from multiple operating system platforms. The use of the *attrib.exe* executable in the enhanced metadata reporting limits this approach. Finally, as a possible extension, the potential use of Portable Python in conjunction with *shadowcopy.py* could provide even more flexibility and rationale for using *shadowcopy.py* on removable media and in Incident Responder's toolkits.

IX. Appendix A

Section II provides an initial overview of VSS technology, including the underlying services, the Previous Versions UI, the creation and management of VSCs, and methods used for rendering VSCs. This section expounds on several aspects introduced in Section II by delving further into the VSS components and the interactions they have with other Microsoft Windows 7 Operating System and third party components as well as the purpose of the VSS artifacts found on a typical system.

A.1. VSS Components and Interactions

An initial overview of the services and driver responsible for the creation and management of VSCs was presented in Section II via introduction of volsnap.sys, the VSS driver, swprv.dll, an intermediary service, and vssvc.exe, the high-level VSS service. This subsection provides additional detail regarding the functionality of these VSS services and the VSS driver. Then, it expounds on the interactions between the VSS services and other components, such as requesters, writers, and providers, which all work together for seamless VSC creation. Next, it discusses various backup methodologies, such as the copy-on-write, complete copy, and redirect-on-write, employed in support of VSC creation. Finally, it ties the various components and interactions together by reiterating Microsoft's 10-step process used to create VSCs.

A discussion of the VSS services, vssvc.exe and swprv.dll, and the VSS driver, volsnap.sys is a good starting point. Volsnap.sys exists on the Windows 7 platform (as well as several other platforms) as a kernel-mode driver that operates "both above and below" the Microsoft Windows NTFS file system.[10] Expanding on this point, volume

snapshot technology operates below the file system, at the block level; however, Volsnap.sys must allocate shadow storage volumes as logical files, so it also operates above the NTFS file system in order to help create logical VSC files in the *System Volume Information* folder.[26] Volsnap.sys is located in the C:\Windows\System32\drivers folder and is responsible for saving the previous versions of a particular block via the copy-on-write functionality as a change will occur to the overlying file/folder.

Swprv.dll exists on the Windows 7 platform (as well as several other platforms) as the VSS software shadow provider. Shadow providers are system, software, or hardware components that serve as interfaces to the point-in-time imaging capabilities.[45] In simpler terms, shadow providers facilitate the creation of shadow copies by managing running volumes and are responsible for on-demand creation of shadow copies from the running volumes.[45] Microsoft designed the VSS architecture to allow other, third party, VSS providers to exist at the intermediate service level at which swprv.dll operates. Other VSS providers may provide alternate functionality for the interaction between the volsnap.sys driver and higher-level VSS service(s).

In addition to being facilitated through the VSS service component, VSS activities also require the use of shadow requestors, writers, and as previously discussed, native and third party system, hardware, and software providers. Figure 30 depicts this relationship.[Credit to Microsoft]

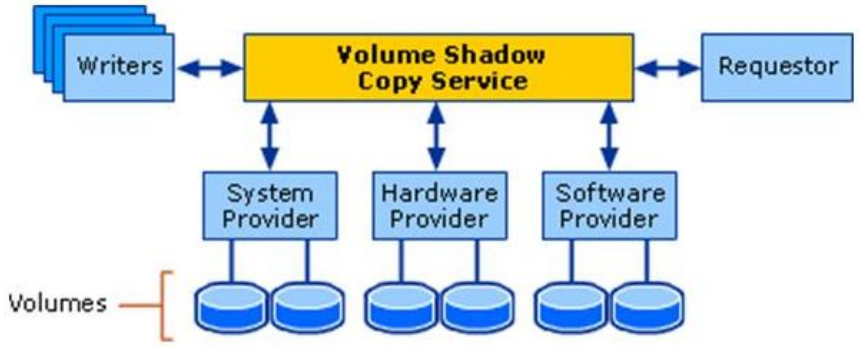


Figure 30: Relationship of VSS to Writers, Requestors, and Providers

As depicted above, shadow requestors are backup applications that invoke the VSS service, which then communicates with providers and writers to perform backup functions. In short, they initiate shadow copy creation as well as other functions. DiskShadow, the Windows shadow requestor, offers both interactive (command-line) and scriptable modes for controlling the VSS service.[46] Shadow writers are application components in Windows applications that help prevent data inconsistencies and provide consistent shadow copies by:

1. communicating with the VSS interface so that applications can prepare and quiesce their data stores, and
2. communicating application information (name, icons, included files, excluded files, and restore strategy).[45]

The native Windows Operating System VSS provider, Swprv.dll, employs a copy-on-write methodology. Additional methodologies include the complete copy and the redirect-on-write, both of which are discussed after the copy-on-write methodology.

A copy-on-write (also known as an incremental copy) is a methodology which preserves original state by reading each block that will be modified and writing it to the VSC just prior to a write input/output operation updating the block's state on the original volume.

VSCs produced by the copy-on-write methodology are also termed as bit level incremental backups of a volume.[7] Terminology refers to the "diff area," which for Windows VSS providers refers to the location where the data for the shadow copy that is created by the system software provider is stored. This diff area can be located on any local NTFS volume with enough space to store it. VSS may create shadow copies of non-NTFS volumes; however, persistent shadow copies, or those that persist across reboots, "can be made only for NTFS volumes. In addition, at least one volume mounted on the system must be an NTFS volume."[47] If accessed through Windows Explorer, VSCs appear as read-only shares.[10] Figure 31, an adaptation of Crabtree [9], depicts a simplified version of an incremental copy.

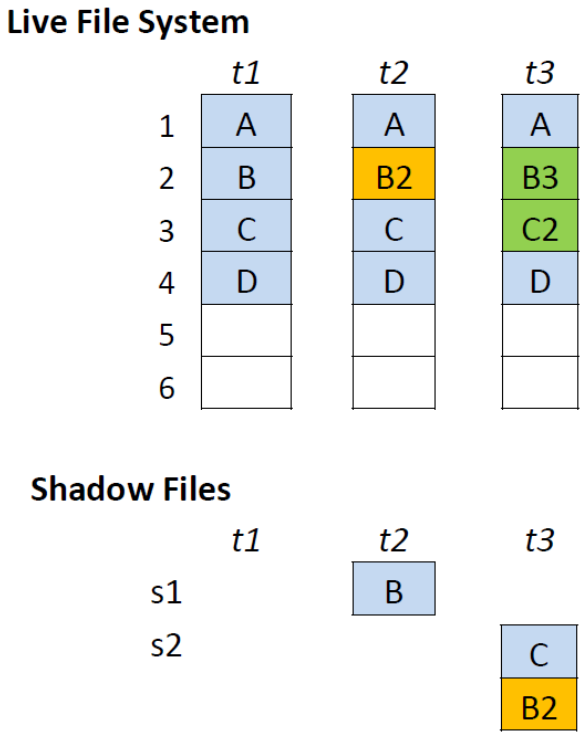


Figure 31: Simplified version of the incremental copy concept

As depicted above, during the 1st time period, or *t1*, only original data A-D exist. During the 2nd time period, or *t2*, just prior to B being overwritten by B2, the original B contents

are preserved via copy-on-write technology and are thus written to the 1st VSC, or *s1*. During the 3rd time period, or *t3*, just prior to B2 being overwritten by B3 and C being overwritten by C2, the original B2 and C contents are preserved via copy-on-write technology and are thus written to the 2nd VSC, or *s2*.

In contrast to the copy-on-write methodology described and depicted above, a complete copy (also known as a full copy, a clone, or a split mirror), is a full duplicate of the original data, created by a software or hardware provider. The clone remains synchronized until the mirror connection is broken. The *live* volume continues to be written-to while the shadow copy remains a read-only version of the *live* volume's state at the exact instant the connection was broken.[45]

A redirect-on-write is somewhat similar to the copy-on-write methodology, however, instead of writing changes to the original volume, it preserves the original state on the original volume and writes incremental changes to the VSC.[47] Figure 32, an adaptation of Crabtree [9], depicts a simplified version of a redirect-on-write copy.

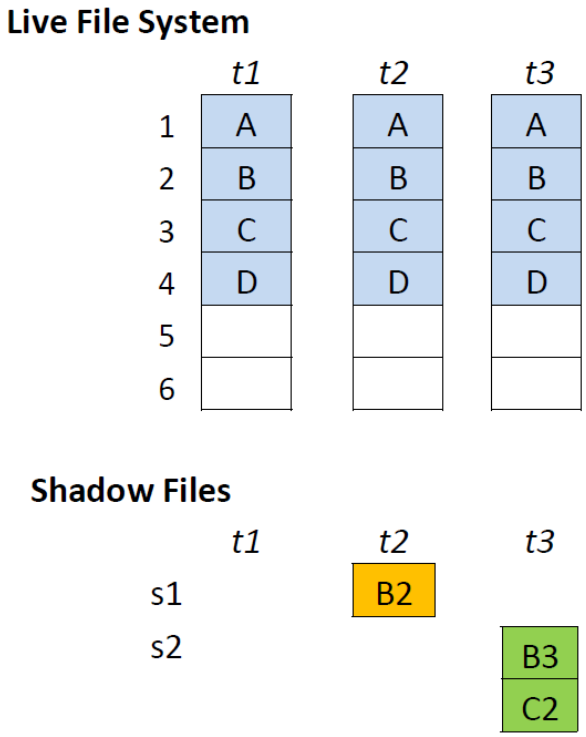


Figure 32: Simplified version of the redirect-on-write copy concept

Regardless of whether the methodology employed is copy-on-write, redirect-on-write, or complete copy, the shadow copy creation process is similar. The following figure, Figure 33, and subsequent description, provide an overview of the Shadow Copy Creation Process: (Credit to Microsoft)

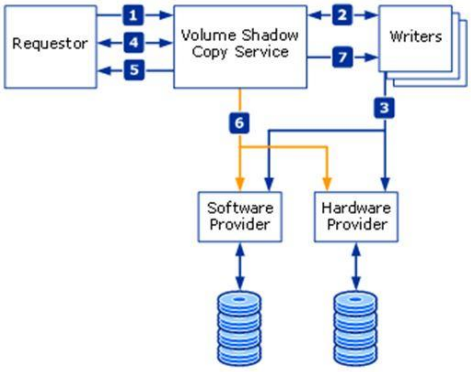


Figure 33: Shadow Copy Creation Process

To create a shadow copy, the requester, writer, and provider perform the following:

1. The requester asks the Volume Shadow Copy Service to enumerate the writers, gather the writer metadata, and prepare for shadow copy creation.

2. Each writer creates an XML description of the components and data stores that need to be backed up and provides it to the Volume Shadow Copy Service. The writer also defines a restore method, which is used for all components. The Volume Shadow Copy Service provides the writer's description to the requester, which selects the components that will be backed up.

3. The Volume Shadow Copy Service notifies all the writers to prepare their data for making a shadow copy.

4. Each writer prepares the data as appropriate, such as completing all open transactions, rolling transaction logs, and flushing caches. When the data is ready to be shadow-copied, the writer notifies the Volume Shadow Copy Service.

5. The Volume Shadow Copy Service tells the writers to temporarily freeze application write I/O requests (read I/O requests are still possible) for the few seconds that are required to create the shadow copy of the volume or volumes. The application freeze is not allowed to take longer than 60 seconds. The Volume Shadow Copy Service flushes the file system buffers and then freezes the file system, which ensures that the file system metadata is recorded

correctly and the data to be shadow-copied is written in a consistent order.

6. The Volume Shadow Copy Service tells the provider to create the shadow copy. The shadow copy creation period lasts no more than 10 seconds, during which all write I/O requests to the file system remain frozen.

7. The Volume Shadow Copy Service releases file system write I/O requests.

8. VSS tells the writers to thaw application write I/O requests. At this point applications are free to resume writing data to the disk that is being shadow-copied.

9. The requester can retry the process (go back to step 1) or notify the administrator to retry at a later time.

10. If the shadow copy is successfully created, the Volume Shadow Copy Service returns the location information for the shadow copy to the requester. In some cases, the shadow copy can be temporarily made available as a read-write volume so that VSS and one or more applications can alter the contents of the shadow copy before the shadow copy is finished. After VSS and the applications make their alterations, the shadow copy is made read-only. This phase is called Auto-recovery, and it is used to undo any file-system or application transactions on the shadow copy volume that were not completed before the shadow copy was created.[47]

An additional VSS feature noted by Mark McKinnon is that a single arbitrary VSC may store more than one change to the same source data blocks. While unconfirmed, it is speculated this feature was created as a mechanism to record minor *intermediate* changes without taking on the overhead of an entire new incremental copy. This discovery created additional research for McKinnon/Whitfield and ultimately led to a parsing methodology, discussed briefly in Section III and subsection B.5 of Appendix B, as well as added functionality to be incorporated into the Shadow Analyser utility, discussed briefly in subsection C.4 of Appendix C, to accommodate for it.

This subsection discussed the functionality of the VSS services and the VSS driver as well as the interactions between the VSS services, requesters, writers, and providers. It discussed the copy-on-write, complete copy, and redirect-on-write methods. Finally, it reiterated Microsoft's 10-step process used to create VSCs. The following subsection provides additional detail into the attributes and artifacts used on a system to manage VSS and the system's VSCs.

A.2. VSS Attributes and Artifacts

By default, 5% of a disk is reserved for the VSS service. The reserved size is configurable and may be increased arbitrarily. Should the VSCs arrive at consuming the 5% or configured capacity of disk space, a pruning mechanism removes older VSCs in a FIFO capacity. In order to conserve disk space, temporary files such as paging files are automatically omitted from shadow copies.[47]

To make VSC writing more efficient, Volsnap.sys initially pre-allocates 600MB of space for shadow storage. This occurs for three reasons:

1. to prevent deadlock when trying to write to the VSC (since the VSC space must immediately be available for writes and it must receive writes prior to the active file system receiving writes),

2. to avoid file growth restrictions during shadow copy creation ("NTFS write I/O is essentially blocked for … the 'flush-and-hold' interval"), and

3. to ensure the VSC is available at the earliest opportunity for copy-on-write of "hot blocks."[26]

The Windows 7 Operating System has several registry keys that control various aspects of VSS and VSCs, to include the minimum initial disk space reserved for the VSS service as well as files that should not be backed-up.[47] Table 5 lists the documented VSS registry artifacts and provides descriptive information regarding the usage of the artifacts.

Key/Hive	**Usage**	**Notes**
VssAccessControl (Key)	Specifies which users have access to shadow copies.	Default is NT Authority\NetworkService. Key is located in Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet[and ControlSet001]\services\VSS.
MaxShadowCopies (Key)	Specifies the limit for the Shadow Copies of Shared Folders feature. (Default is 64.)	The maximum number of client-accessible software shadow copies that can be stored on each volume of the computer is 512.
MinDiffAreaFileSize (Key)	Specifies the minimum initial size, in MB, of the shadow copy storage area.	600MB of shadow copy storage area is pre-allocated by Volsnap.sys.
FilesNotToSnapshot (Key)	Specifies which files to exclude from shadow copies.	It **cannot** delete files from a shadow copy that was created on a Windows Server by using the Previous Versions feature. It **cannot** delete files from shadow copies for shared folders. It **can** delete files from a shadow copy that was created by using the Diskshadow utility, but it **cannot** delete files from a shadow copy that was created by using the Vssadmin utility.

		Files are deleted from a shadow copy on a best-effort basis. This means that they are not guaranteed to be deleted.
\System Volume Information\Syscache.hve (Hive)		New registry hive.

Table 5: VSS Registry Artifacts

X. Appendix B

Sections III and IV provide high-level overviews and analysis of several methods commonly used in support of digital investigations. Those include, but are not limited to, using the Windows Previous Versions UI, *vssadmin* and *mklink*, *vssadmin* and *net share*, restoring and accessing, parsing VSCs, *fls* and *mactime*, and specialized utilities/methods. Each subsection of this Appendix provides additional details, which were relevant to the analysis of each of the aforementioned tools/techniques, in an unstructured format. First is a use case for the Windows Previous Versions UI.

B.1. Using the Windows Previous Versions UI

The UI may be accessed by right-clicking on an arbitrary file or folder and then selecting the "Restore previous versions" dialogue from the menu. Next, the item desired is selected and the "Restore" option is selected. Figure 34 depicts this methodology.

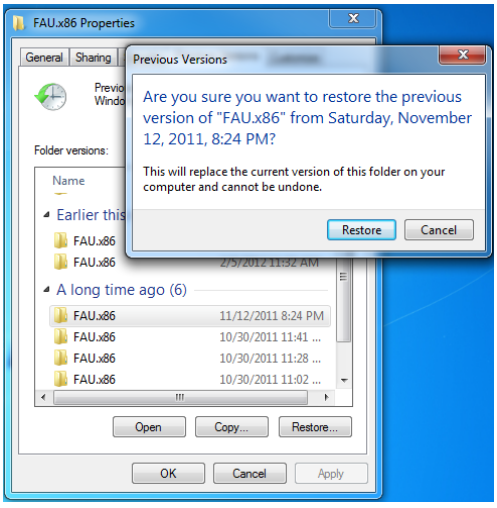


Figure 34: Previous Versions UI "Restore" option dialogue

An alternate methodology for restoring a previous version of an item without overwriting the current version of that item is to select the item, but instead of selecting the "Restore"

option, drag and drop the item to another location such as the Desktop. Figure 35 depicts this methodology.

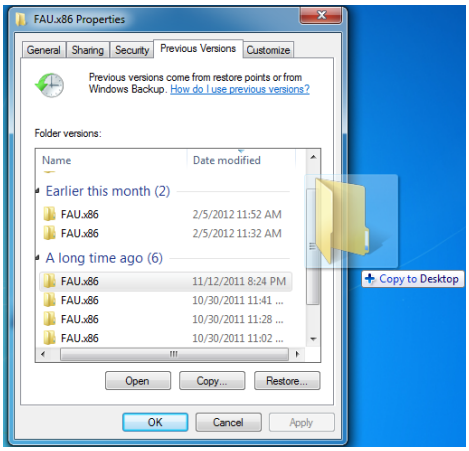


Figure 35: Previous Versions UI copy restoration methodology

After providing a very brief description of the use case for the Windows Previous Versions UI, the *vssadmin* and *mklink* method is addressed, next.

B.2. Using *vssadmin* and *mklink*

The use case for the *vssadmin* and *mklink* method is as follows:

1. First, mount the image of the disk or partition containing the VSCs.

2. Next, add the mounted image as a new disk to a VMWare guest that is loaded with the Windows 7 Professional Operating System. Ensure the "use a physical disk (for advanced users)" option is selected. Note: If mounting a Virtual Hard Disk (VHD) format disk image file versus a raw (*dd*) format disk image file, one may skip Step #1 and must use the “use an existing virtual disk” option instead of "use a physical disk (for advanced users),” then browse to the VHD file, and select “Independent” and “Nonpersistent” mode.

3. Boot the VM and mount the shadow copies with *vssadmin* and *mklink* as follows:

a. `vssadmin list shadows /for=[Volume]:`

b. `mklink /d [Volume]:\rp[Shadow Volume Number]

\\?\GLOBALROOT\ Device\HarddiskVolumeShadowCopy[Number]\`

This executes the aforementioned *mklink* command (with the /d argument) in order to create a directory symbolic link for any arbitrary VSC previously referenced in the *vssadmin* command. After the VSC(s) is mounted and accessible via the directory symbolic link(s), one may selectively view the VSC contents using the Windows command line commands *cd* and *dir*.

4. After analysis of VSC contents is complete, one may perform "cleanup" by executing the *rd* command below or by reverting to a snapshot in the surrogate, or "analysis VM."

a. `rd [Volume]:\rp[Shadow Volume Number]`

This executes the *rd* command (removes (deletes) a directory) in order to remove the directory symbolic link for any arbitrary VSC(s). Afterward, the VSC(s) is unmounted and is no longer accessible via the directory symbolic link(s).

Figures 36 through 40 depict these methodologies. *Note: In the following test case, the surrogate system (native volume) VSCs (not pictured) were numbered 1...8 and the VSCs of the mounted VHD format disk image file (those intended for analysis) were numbered 9...20. This VSC numbering is only temporary, as VSC numbering is specific to the environment in which the *vssadmin* command is executed.

```
C:\Windows\system32>vssadmin list shadows /for=e:
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001-2005 Microsoft Corp.

Contents of shadow copy set ID: {8c30bd38-30b4-47c7-ad91-b06400253f6a}
   Contained 1 shadow copies at creation time: 11/6/2011 8:11:05 PM
      Shadow Copy ID: {1b31f3da-1aac-485a-816f-e80dc029f958}
         Original Volume: (E:)\\?\Volume{4c68be2f-60e8-11e1-bf0c-000c2922f57d}\
         Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy9
         Originating Machine: VSCTest-PC
         Service Machine: VSCTest-PC
         Provider: 'Microsoft Software Shadow Copy provider 1.0'
         Type: ClientAccessibleWriters
         Attributes: Persistent, Client-accessible, No auto release, Differential, Auto recovered
...
Contents of shadow copy set ID: {d1ff5136-a484-4753-b9cc-b70e3df8c46e}
   Contained 1 shadow copies at creation time: 11/9/2011 6:02:18 AM
      Shadow Copy ID: {beb2bc9d-7dff-495e-89d9-467d29144568}
         Original Volume: (E:)\\?\Volume{4c68be2f-60e8-11e1-bf0c-000c2922f57d}\
         Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy20
         Originating Machine: VSCTest-PC
         Service Machine: VSCTest-PC
         Provider: 'Microsoft Software Shadow Copy provider 1.0'
         Type: ClientAccessibleWriters
         Attributes: Persistent, Client-accessible, No auto release, Differential, Auto recovered
```

Figure 36: Executing the *vssadmin list shadows* command

```
C:\Windows\system32>mklink /d c:\rp9 \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy9\
symbolic link created for c:\rp9 <<===>> \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy9\

...

C:\Windows\system32>mklink /d c:\rp20 \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy20\
symbolic link created for c:\rp20 <<===>> \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy20\
```

Figure 37: Executing the *mklink* command

```
C:\>dir
 Volume in drive C has no label.
 Volume Serial Number is 78B7-4C9A

 Directory of C:\
06/10/2009  04:42 PM            24 autoexec.bat
06/10/2009  04:42 PM            10 config.sys
07/13/2009  09:37 PM    <DIR>      PerfLogs
10/30/2011  09:14 PM    <DIR>      Program Files
02/26/2012  09:43 PM    <SYMLINKD>     rp10 [\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy10\]
...
02/26/2012  09:43 PM    <SYMLINKD>     rp20 [\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy20\]
02/26/2012  09:43 PM    <SYMLINKD>     rp9 [\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy9\]
10/09/2011  09:25 AM    <DIR>      Users
10/10/2011  02:41 PM    <DIR>      Windows
               2 File(s)             34 bytes
              16 Dir(s)   6,874,152,960 bytes free
```

Figure 38: Executing the *DIR* command to show symlinks

```
C:\>dir rp9\*
 Volume in drive C has no label.
 Volume Serial Number is 78B7-4C9A

 Directory of C:\rp9

07/13/2009  10:20 PM    <DIR>          PerfLogs
07/14/2009  02:47 AM    <DIR>          Program Files
07/13/2009  11:57 PM    <DIR>          Program Files (x86)
11/06/2011  08:07 PM    <DIR>          Users
11/07/2011  01:53 AM    <DIR>          Windows
               0 File(s)              0 bytes
               5 Dir(s)  29,727,424,512 bytes free

C:\>dir rp20\*
 Volume in drive C has no label.
 Volume Serial Number is 78B7-4C9A

 Directory of C:\rp20

07/13/2009  10:20 PM    <DIR>          PerfLogs
11/08/2011  06:51 AM    <DIR>          Program Files
11/06/2011  11:18 PM    <DIR>          Program Files (x86)
11/06/2011  08:07 PM    <DIR>          Users
11/09/2011  05:54 AM    <DIR>          Windows
               0 File(s)              0 bytes
               5 Dir(s)  16,262,864,896 bytes free
```

Figure 39: *DIR* commands showing differences in VSCs #9 and #20

```
C:\>rd c:\rp9

...

C:\>rd c:\rp20

C:\>dir
Volume in drive C has no label.
Volume Serial Number is 78B7-4C9A

Directory of C:\

06/10/2009  04:42 PM            24 autoexec.bat
06/10/2009  04:42 PM            10 config.sys
07/13/2009  09:37 PM    <DIR>          PerfLogs
10/30/2011  09:14 PM    <DIR>          Program Files
10/09/2011  09:25 AM    <DIR>          Users
10/10/2011  02:41 PM    <DIR>          Windows
2 File(s)             34 bytes
4 Dir(s)   6,874,128,384 bytes free
```

Figure 40: *RD* command removes symbolic directory links

After providing a very brief description of the use case for the *vssadmin* and *mklink* method, the *vssadmin* and *net share* method are addressed, next.

B.3. Using *vssadmin* and *net share*

The use case for the *vssadmin* and *net share* method is as follows:

1. First, the command *vssadmin list shadows /for=[Volume]:* is issued to provide a listing of all shadows available for the volume specified.

2. Next, the command *net share testshadow=\\.\HarddiskVolumeShadowCopy [Shadow Volume Number]* is issued to mount an arbitrary VSC under the share name *testshadow*. After one or more VSC(s) is mounted and becomes accessible via the share(s), one may selectively view the contents using the command-line commands *cd* and *dir*.

3. After analysis of VSC contents is complete, one may perform "cleanup" by running the *net share {share} /DELETE* command below or by reverting to a snapshot in the surrogate, or "analysis VM."

"net share testshadow[Number] /DELETE"

This executes the *net share /DELETE* command (removes (deletes) a network share) in order to remove the network share for any arbitrary VSC(s). After executing the command, the VSC(s) is unmounted and is no longer accessible via the network share(s).

Figures 41 through 44 depict these methodologies. Figure 44 provides the best depiction of the end result a digital investigator sees when using this method – all VSCs are mounted as *testshadow[x]* within Windows Explorer.

```
C:\>net share testshadow9=\\.\HarddiskVolumeShadowCopy9\
testshadow9 was shared successfully.

...

C:\>net share testshadow20=\\.\HarddiskVolumeShadowCopy20\
testshadow20 was shared successfully.
```

Figure 41: Executing the *net share <VSC>* command

```
C:\Windows\system32>net share                (Before)

Share name   Resource                Remark
------------------------------------------------------------------------------
C$           C:\                     Default share
E$           E:\                     Default share
F$           F:\                     Default share
IPC$                                 Remote IPC
ADMIN$       C:\Windows              Remote Admin
The command completed successfully.

C:\Windows\system32>net share                (After)

Share name   Resource                Remark
------------------------------------------------------------------------------
C$           C:\                     Default share
E$           E:\                     Default share
F$           F:\                     Default share
IPC$                                 Remote IPC
ADMIN$       C:\Windows              Remote Admin
testshadow10 \\.\HarddiskVolumeShadowCopy...
...
testshadow20 \\.\HarddiskVolumeShadowCopy...
testshadow9  \\.\HarddiskVolumeShadowCopy9\
The command completed successfully.
```

Figure 42: Windows shares before and after the VSCs are mounted as "testshadowX"

```
C:\>net share testshadow9 /DELETE
testshadow9 was deleted successfully.

...

C:\>net share testshadow20 /DELETE
testshadow20 was deleted successfully.
```

Figure 43: Methodology for removing the Windows shares

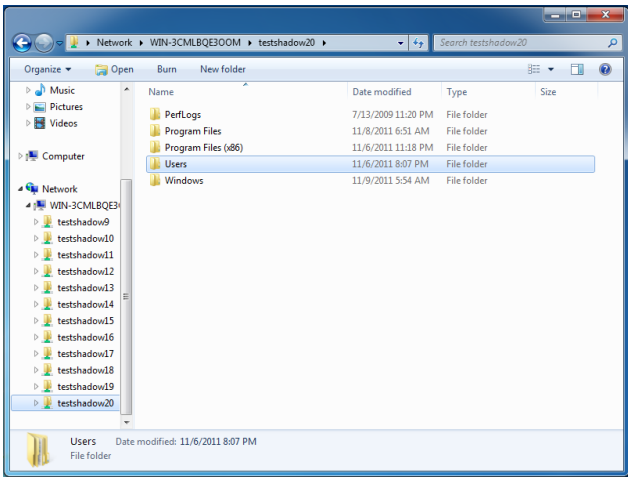


Figure 44: Mounted VSCs, now accessible via Windows shares

After providing a very brief description of the use case for the *vssadmin* and *net share* method, the restoring and accessing method is addressed, next.

B.4. Using restoring and accessing

The use case for the restoring and accessing method is as follows:

Lee [19], Carvey [48], and "DC1743" [41], describe mounting and file conversion methods using freeware tools, such as LiveView, VMWare VDDK 1.2 (vmware-mount), VHDTOOL.exe, and Windows 7's native Disk Manager, which are pre-requisites for preparing an original drive/partition image for the subsequent VSC imaging process. Lee discusses the methodology for converting a raw (*dd* format) disk image file into a Virtual Machine Disk (VMDK) file using the LiveView utility.[18] Carvey discuss manually mounting and accessing disk/partition files in two different ways: as a VMDK file [20] or as a Virtual Hard Disk (VHD) file [24]. If using the VMDK method, then using the

vmware-mount command, *vmware-mount /p*, one can see all partitions of a drive image as well as imaged shadowed volumes within a virtual disk file. Figure 45 depicts this.

```
C:\Program Files (x86)\VMware\VMware Virtual Disk Development Kit\bin>vmware-mount
/p "D:\Thesis DD Images for Analysis\VSCTest-PC_12Nov11_Liveview VMDK Files\VSCTest-
PC_12Nov11.dd.vmdk" (Drive image containing two partitions)
Volume  1 :     100 MB, HPFS/NTFS
Volume  2 :    38064 MB, HPFS/NTFS

C:\Program Files (x86)\VMware\VMware Virtual Disk Development Kit\bin>vmware-mount
/p "D:\Thesis Binaries\Liveview\d_19Feb12_image.img.vmdk" (imaged Shadow Copy)
Volume  1 :    95386 MB, HPFS/NTFS
```

Figure 45: *Vmware-mount* command demonstrating standard partitions and VSC container

Using the following approach, Microsoft's Virtual Hard Disk conversion tool (*vhdtool.exe*) and the Windows Operating System may be used to convert a raw format (*dd*) disk image file to a VHD file and then mount that file:

1. Execute *vhdtool.exe /convert* against the *dd* format disk image file
2. Open the Computer Management interface in Windows 7 followed by the Disk Manager
3. Select *Action -> Attach VHD*, ensuring the "Read-only" box is checked, and then select "ok."

This causes the disk and the volume listing to be visible in the Disk Manager. The drive icon should appear light blue in color (representing a VHD) versus the standard grey in color (physical Hard Disk) icons for other drives.

Lee [19], "DC1743" [49], Carvey [48], and "ecophobia" [50] discuss the next step: using George Garner's data dump utility, *dd.exe*, from the Forensic Acquisition Utilities, to image the VSC. The methodology to image a shadow copy to a "flat" file using *dd.exe* may be accomplished as follows: *dd.exe if=\\.\HarddiskVolumeShadowCopy[shadow volume number] of=[LOGICAL DRIVE LETTER]:\snapshot[shadow volume*

number].img --localwrt. The parameters given are the "if," or input file, and the "of," or output file. The input file is set to point to the VSC the user wants to image. The output file is set to point to a flat file on a logical volume. The *localwrt* argument causes the *dd.exe* command to write to a locally mounted drive. The *dd.exe* methodology is depicted below in Figure 46.

```
D:\Thesis Binaries\FAU (Garner)\fau\FAU.x64>dd
if=\\.\HarddiskVolumeShadowCopy1
of=d:\d_19Feb12_image.img --localwrt
The VistaFirewall Firewall is active with exceptions.

Copying \\.\HarddiskVolumeShadowCopy1 to
d:\d_19Feb12_image.img
Output: d:\d_19Feb12_image.img
100019466240 bytes
95385+1 records in
95385+1 records out
100019466240 bytes written

Succeeded!
```

Figure 46: *DD.exe* methodology for imaging a VSC

After imaging is complete, mounting the VSC is the next process. Lee demonstrates using the *ntfs-3g —o ro,loop,show_sys_files snapshot[shadow volume number].img /PATH/snapshot[shadow volume number]* command to mount the VSC.[19] The VSC may also be mounted via the *mklink*, *net share*, and other discussed methods.

After providing a very brief description of the use case for the restoring and accessing method, the (non API-restrictive) VSC parsing method is addressed, next.

B.5. Parsing VSCs

At a high level, the VSC parsing methodology consists of the following:

1. Pre-processing (preparing the database which will store parsed information and parsing the master boot record and partitions of the drive image to be analyzed),
2. Parsing the $MFT,
3. Gathering the VSC information,
4. Parsing the VSCs from newest to oldest, and
5. Reporting.

At a high level, the $MFT parsing subcomponent consists of the following:

1. Reading the first $MFT record and obtaining the size of the $MFT file,
2. Parsing all $MFT records,
    a. Writing all parsed data to database entries,
    b. Determining the run size for non-resident entries and processing each run. Using the $DATA attribute to determine whether the data is resident in the $MFT or in a run list.
3. Recreating the directory structure in a database table, and
4. Creating the $MFT record lookup data structure.

At a high level, the VSC parsing subcomponent consists of the following:

1. Opening the VSC,
2. Reading the VS header block and storing its information in the database,
3. Processing each 16KB block.
    a. If the block is an Index block, then parsing as such,
    b. If the block is an $MFT record block, then parsing as such, and

Reading the $MFT entries from oldest to newest (order is reversed such that if a record is newer, its contents are retrieved; if a record is not newer, then one has the current data).

Courtesy of McKinnon, Figures 47 through 49 depict the processes involved in the VSC Parsing methodology, the processes for parsing the $MFT, and the processes for parsing VSCs, respectively.[29]

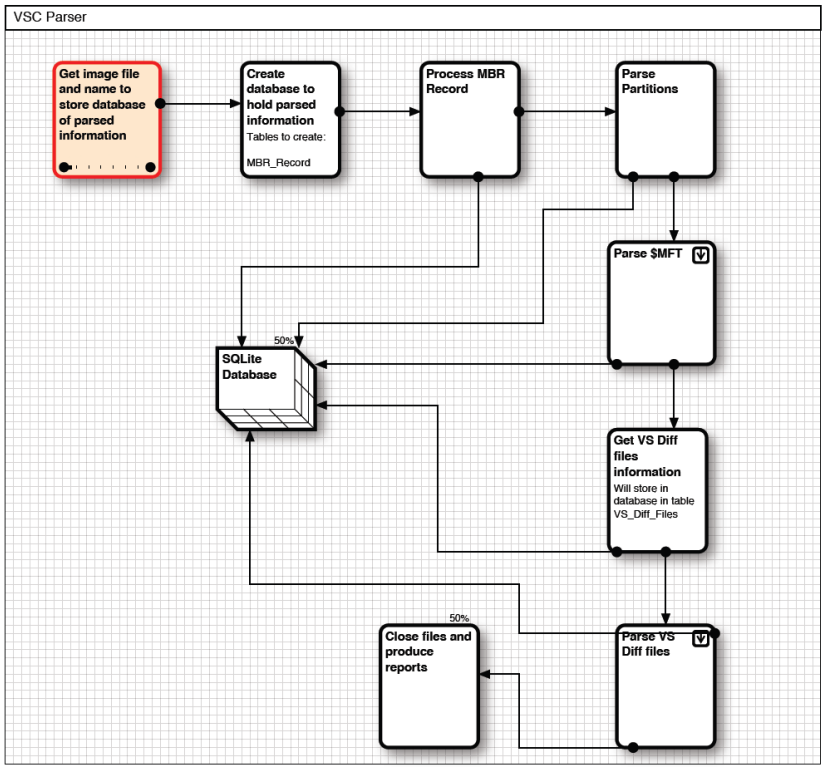


Figure 47: VSC Parser Process Flow

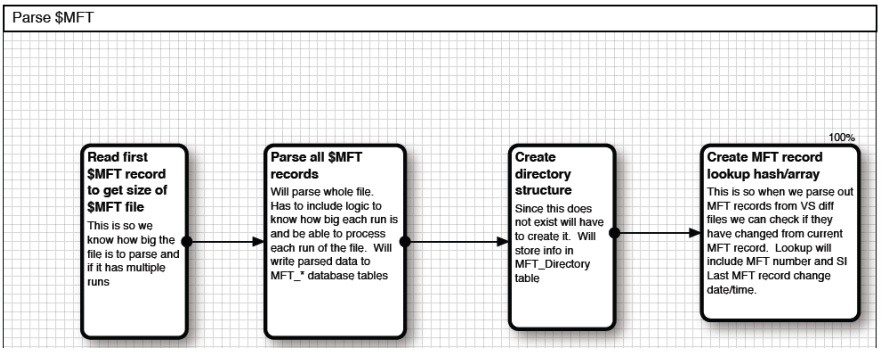


Figure 48: Parse $MFT Process Flow

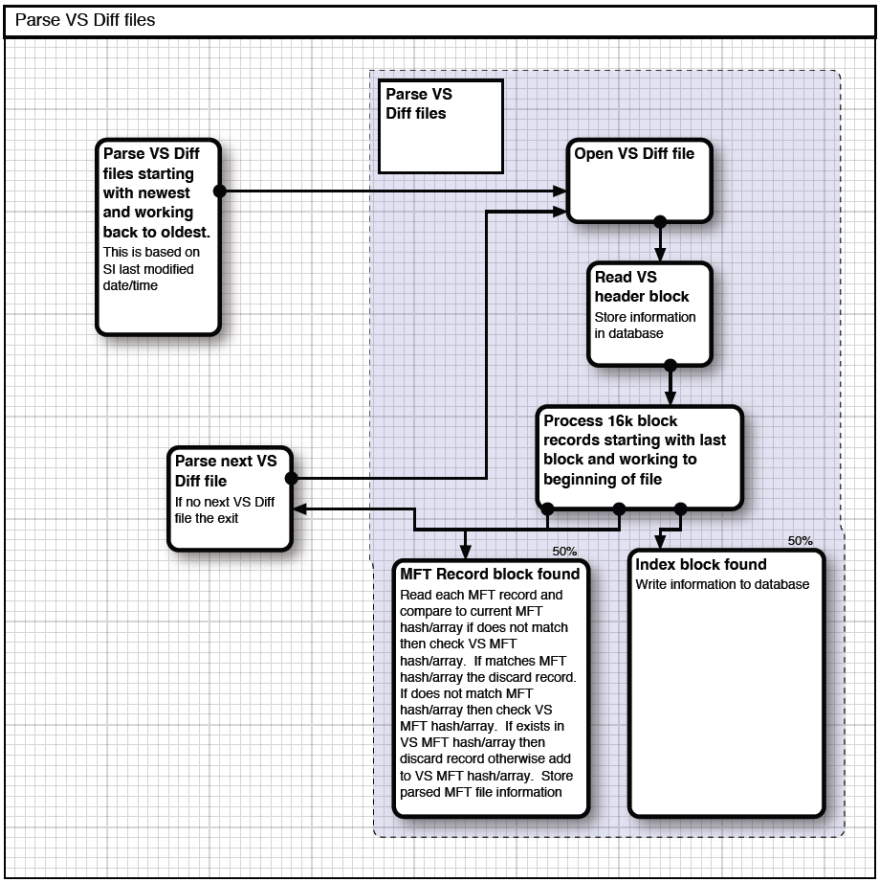


Figure 49: Parse VS Diff Files Process Flow

After providing a very brief description of the use case for the VSC parsing method, the *fls* and *mactime* method is addressed, next.

B.6. Using *fls* and *mactime*

The *fls* and *mactime* approach requires that the digital investigator mount the disk image file containing the VSCs and *live* volume using the Microsoft Windows 7 Computer Management Interface (including the corresponding Disk Manager element) or another utility. The process relies on the Windows disk class driver, volume manager driver, partition manager, I/O manager, CreateFile function, and VSS API to facilitate access to the disk image file as well as the *live* volume and VSCs contained therein.[8] The VSCs are accessed as disk device objects using the device object nomenclature,

"\\.\HarddiskVolumeShadowCopy[shadow volume number]," which is similar to the nomenclature used to access standard disk volumes, "\\.\HarddiskVolume[number]." After mounting the disk image file, the use case for the *fls* and *mactime* method is as follows:

1. First, use *fls* to extract bodyfile info:

   The normal use of fls for extracting bodyfile info from a traditional partition is: *fls -r -m [Drive]: \\.\[Drive]: > \\WORKSTATION\ShadowTime\bodyfile*

   In order to extract bodyfile info from a ShadowVolume, use the command:

   fls -r -m Shadow[shadow volume number]/ \\.\HarddiskVolumeShadowCopy [shadow volume number] >> \\WORKSTATION\ShadowTime\bodyfile

   fls lists the files and directory names in the VSC. The *-r* argument forces it to recursively display directories and the *-m mnt* argument forces it to display files in "time machine format" so that a timeline can be created with *mactime*.

2. Next, run *mactime* to dump the bodyfile into a timeline in CSV format:

   mactime -d -b bodyfile > shadow_timeline.csv

   mactime "creates an ASCII timeline of file activity" based on *fls'* output by importing the data from the body file, sorting that data, and printing the output. The *–d* argument forces it to format output in comma delimited format and the *–b* argument specifies the bodyfile name.

A modification of this method provides all the results compiled into one bodyfile using the same *–m mnt* argument (instead of a separate argument for each arbitrary partition and/or VSC) and thus enhances one's ability sort uniquely, eliminating duplicate entries. This methodology does not retain the ability to determine which of the VSCs contained

any metadata of concern. It may, however, be a viable method for extracting all non-unique metadata in support of the visualization of change over time.

Figure 50 depicts the syntax to validate the methodology using multiple bodyfiles:

```
D:\Thesis Binaries\TSK\sleuthkit-win32-3.2.3\sleuthkit-win32-3.2.3\bin>fls -r -m
 Shadow5/ \\.\HarddiskVolumeShadowCopy60 > "d:\bodyfile - VSC5(60)_only"
D:\Thesis Binaries\TSK\sleuthkit-win32-3.2.3\sleuthkit-win32-3.2.3\bin>fls -r -m
 Shadow10/ \\.\HarddiskVolumeShadowCopy65 > "d:\bodyfile - VSC10(65)_only"

D:\Thesis Binaries\TSK\sleuthkit-win32-3.2.3\sleuthkit-win32-3.2.3\bin>mactime.p
l -d -b "d:\bodyfile - VSC5(60)_only" > d:\shadow_timeline_separate.csv
D:\Thesis Binaries\TSK\sleuthkit-win32-3.2.3\sleuthkit-win32-3.2.3\bin>mactime.p
l -d -b "d:\bodyfile - VSC10(65)_only" >> d:\shadow_timeline_separate.csv
```

Figure 50: *fls* and *mactime* syntax for exporting VSC timelines

Table 6 depicts the final output after importing the results into Microsoft Excel and filtering results showing the NTUser.dat file for the user VSCTest:

Date	Size	Type	Mode	Meta	File Name
Sun Nov 06 2011 20:07:28	524288	...b	r/rr-xr-xr-x	511-128-1	Shadow5/Users/VSCTest/NTUSER.DAT
Sun Nov 06 2011 22:32:39	524288	.a..	r/rr-xr-xr-x	511-128-1	Shadow5/Users/VSCTest/NTUSER.DAT
Sun Nov 06 2011 22:52:57	524288	m.c.	r/rr-xr-xr-x	511-128-1	Shadow5/Users/VSCTest/NTUSER.DAT
Sun Nov 06 2011 20:07:28	524288	...b	r/rr-xr-xr-x	511-128-1	Shadow10/Users/VSCTest/NTUSER.DAT
Tue Nov 08 2011 05:52:41	524288	.a..	r/rr-xr-xr-x	511-128-1	Shadow10/Users/VSCTest/NTUSER.DAT
Tue Nov 08 2011 06:28:18	524288	m.c.	r/rr-xr-xr-x	511-128-1	Shadow10/Users/VSCTest/NTUSER.DAT

Table 6: Timestamp-formatted Microsoft Excel depiction of VSC metadata using multiple bodyfiles

The results show that the file record was originally created on November 6th 2011 at 20:07. It was subsequently modified at 22:52 and also on November 8th 2011 at 06:28.

Table 7 (below) depicts the final output after validating the results using the single bodyfile methodology (all results combined in one file with the same *–m mnt* argument, *Shadow*), importing the results into Microsoft Excel, and filtering results showing the NTUser.dat file for the user VSCTest:

Date	Size	Type	Mode	Meta	File Name
Sun Nov 06 2011 20:07:28	524288	...b	r/rr-xr-xr-x	511-128-1	Shadow/Users/VSCTest/NTUSER.DAT

Sun Nov 06 2011 22:32:39	524288	.a..	r/rr-xr-xr-x	511-128-1	Shadow/Users/VSCTest/NTUSER.DAT
Sun Nov 06 2011 22:52:57	524288	m.c.	r/rr-xr-xr-x	511-128-1	Shadow/Users/VSCTest/NTUSER.DAT
Tue Nov 08 2011 05:52:41	524288	.a..	r/rr-xr-xr-x	511-128-1	Shadow/Users/VSCTest/NTUSER.DAT
Tue Nov 08 2011 06:28:18	524288	m.c.	r/rr-xr-xr-x	511-128-1	Shadow/Users/VSCTest/NTUSER.DAT

Table 7: Timestamp-formatted Microsoft Excel depiction of VSC metadata using a single bodyfile

This method loses the VSC association to the metadata, however, maintains a unique timeline without additional utilities. Increasing the scope of this method to a larger scale would be necessary to validate whether it could solve the problems associated with extracting all data for all VSCs on a particular system.

After providing a very brief description of the use case for the *fls* and *mactime* method, using "specialized" utilities/methods is addressed, next.

B.7. Using specialized utilities/methods

The use case for specialized utilities/methods (also, arbitrary or "other" methods) is as follows:

"The UserAssist registry key resides in the NTUSER.DAT file on disk at Software\Microsoft\Windows\ CurrentVersion\ Explorer\UserAssist or, in the *live* registry, at HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\ UserAssist."[51] The following methodology is used to extract data from the UserAssist registry key.

1. Use RegRipper to access and extract a UserAssist registry key to determine shortcuts to applications most frequently used on the system:

   rip.exe –p userassist –r [Drive]:\VSC[shadow volume number]\ > user_userassist[shadow volume number].txt

The *–p* argument specifies the RegRipper plugin to execute and the *–r* argument specifies the location of the target VSC.

XI. Appendix C

Sections III and IV provide high-level overviews and analysis of several methods commonly used in support of digital investigations. Those include, but are not limited to, scripting manual tools, *Robocopy*, and *LogParser*. Each subsection of this Appendix provides additional details, which were relevant to the analysis of each of the aforementioned tools/techniques, in an unstructured format. *Shadow Analyser* is an additional untested method that is briefly discussed. Discussed first is a use case for scripting manual tools.

C.1. Scripting manual tools

The methodology is described as follows:

1. First, mount the disk image (*dd*, VHD, or VMDK format) using a drive/partition mounting utility, the Windows 7 Professional Operating System, or a surrogate system.

2. Next, the command *vssadmin list shadows /for=[Volume]:* is issued to provide a listing of all shadows available for the volume specified.

3. Next, issue the following command: `for /l %[Number] in (start,1,stop) do mklink /d c:\rp%[Number] \\?\GLOBALROOT\Device\HarddiskVolume ShadowCopy%[Shadow Volume Number]\`

This uses a *for* loop to iteratively execute the aforementioned *mklink* command (with the /d argument) in order to create a directory symbolic link for each of the VSCs previously referenced in the *vssadmin* command and provided in the loop via the *start* and *stop* options. After the VSCs are mounted and accessible via the

directory symbolic links, one may selectively view their contents using the Windows command line commands *cd* and *dir*.

4. After analysis of VSC contents is complete, one may perform "cleanup" by running the *rd* command below or by reverting to snapshot in the surrogate, or "analysis VM."

a. `for /l %i in (start,1,stop) do rd c:\rp%i`

The *rd* method's *for* loop iteratively executes the *rd* command in order to remove the directory symbolic link for each of the VSCs. Afterward, the VSCs are unmounted and are no longer accessible via the directory symbolic links.

Figures 51 through 55 depict these methodologies. *Note: In the following arbitrary test case, the surrogate system (native volume) VSCs (not pictured) were numbered 1..8 and the VSCs of the mounted VHD format disk image file (those intended for analysis) were numbered 9..20. Numbering was temporary as VSC numbering is specific to the environment in which the VSSAdmin command executes.

```
C:\Windows\system32>vssadmin list shadows /for=e:
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001-2005 Microsoft Corp.

Contents of shadow copy set ID: {8c30bd38-30b4-47c7-ad91-b06400253f6a}
  Contained 1 shadow copies at creation time: 11/6/2011 8:11:05 PM
    Shadow Copy ID: {1b31f3da-1aac-485a-816f-e80dc029f958}
      Original Volume: (E:)\\?\Volume{4c68be2f-60e8-11e1-bf0c-000c2922f57d}\
      Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy9
      Originating Machine: VSCTest-PC
      Service Machine: VSCTest-PC
      Provider: 'Microsoft Software Shadow Copy provider 1.0'
      Type: ClientAccessibleWriters
      Attributes: Persistent, Client-accessible, No auto release, Differential, Auto recovered
...
Contents of shadow copy set ID: {d1ff5136-a484-4753-b9cc-b70e3df8c46e}
  Contained 1 shadow copies at creation time: 11/9/2011 6:02:18 AM
    Shadow Copy ID: {beb2bc9d-7dff-495e-89d9-467d29144568}
      Original Volume: (E:)\\?\Volume{4c68be2f-60e8-11e1-bf0c-000c2922f57d}\
      Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy20
      Originating Machine: VSCTest-PC
      Service Machine: VSCTest-PC
      Provider: 'Microsoft Software Shadow Copy provider 1.0'
      Type: ClientAccessibleWriters
      Attributes: Persistent. Client-accessible. No auto release. Differential. Auto recovered
```

Figure 51: Executing the *vssadmin list shadows* command

```
C:\Windows\system32>for /l %i in (9,1,20) do mklink /d c:\rp%i \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy%i\

C:\Windows\system32>mklink /d c:\rp9 \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy9\
symbolic link created for c:\rp9 <<===>> \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy9\

…

C:\Windows\system32>mklink /d c:\rp20 \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy20\
symbolic link created for c:\rp20 <<===>> \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy20\
```

Figure 52: Executing the *mklink* command

```
C:\>dir
 Volume in drive C has no label.
 Volume Serial Number is 78B7-4C9A

 Directory of C:\
06/10/2009  04:42 PM            24 autoexec.bat
06/10/2009  04:42 PM            10 config.sys
07/13/2009  09:37 PM    <DIR>       PerfLogs
10/30/2011  09:14 PM    <DIR>       Program Files
02/26/2012  09:43 PM    <SYMLINKD>    rp10 [\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy10\]
...
02/26/2012  09:43 PM    <SYMLINKD>    rp20 [\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy20\]
02/26/2012  09:43 PM    <SYMLINKD>    rp9 [\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy9\]
10/09/2011  09:25 AM    <DIR>       Users
10/10/2011  02:41 PM    <DIR>       Windows
               2 File(s)             34 bytes
              16 Dir(s)   6,874,152,960 bytes free
```

Figure 53: Executing the *DIR* command to show symlinks

```
C:\>dir rp9\*
 Volume in drive C has no label.
 Volume Serial Number is 78B7-4C9A

 Directory of C:\rp9

07/13/2009  10:20 PM    <DIR>       PerfLogs
07/14/2009  02:47 AM    <DIR>       Program Files
07/13/2009  11:57 PM    <DIR>       Program Files (x86)
11/06/2011  08:07 PM    <DIR>       Users
11/07/2011  01:53 AM    <DIR>       Windows
               0 File(s)              0 bytes
               5 Dir(s)  29,727,424,512 bytes free

C:\>dir rp20\*
 Volume in drive C has no label.
 Volume Serial Number is 78B7-4C9A

 Directory of C:\rp20

07/13/2009  10:20 PM    <DIR>       PerfLogs
11/08/2011  06:51 AM    <DIR>       Program Files
11/06/2011  11:18 PM    <DIR>       Program Files (x86)
11/06/2011  08:07 PM    <DIR>       Users
11/09/2011  05:54 AM    <DIR>       Windows
               0 File(s)              0 bytes
               5 Dir(s)  16,262,864,896 bytes free
```

Figure 54: *DIR* commands showing differences in VSCs #9 and #20

```
C:\>for /l %i in (9,1,20) do rd c:\rp%i

C:\>rd c:\rp9

…

C:\>rd c:\rp20

C:\>dir
 Volume in drive C has no label.
 Volume Serial Number is 78B7-4C9A

 Directory of C:\

06/10/2009  04:42 PM            24 autoexec.bat
06/10/2009  04:42 PM            10 config.sys
07/13/2009  09:37 PM    <DIR>       PerfLogs
10/30/2011  09:14 PM    <DIR>       Program Files
10/09/2011  09:25 AM    <DIR>       Users
10/10/2011  02:41 PM    <DIR>       Windows
               2 File(s)             34 bytes
               4 Dir(s)   6,874,128,384 bytes free
```

Figure 55: *RD for* loop removes symbolic directory links

5. Harrell further automates the *for* loop controlled VSC mount/dismount process using *vssadmin*, *mklink*, and *rd* by encapsulating it within a batch script. The script divides the process into three distinct phases by incorporating them into functions *listvsc()*, *makelink()*, and *removelink()*. *Listvsc()* prompts the user for the drive letter upon which to mount VSCs and then executes the *vssadmin list shadows* command. Depending on user input, it produces output solely to the console or to the console as well as to a text file. *Makelink()* prompts a user for the starting and ending VSC volumes and the executes a *for* loop controlled *mklink* command similar to Step #3 (above) to iteratively mount each one. *Removelink()* prompts a user for the starting and ending VSC volumes and the executes a *for* loop controlled *rd* command similar to Step #4 (above) to iteratively dismount each one.[22]

Hargreaves also provides a command string, *vssadmin list shadows /for=c:\ >c:\Restorepoints_on_C.txtfor /f "tokens=4" %%f in ('vssadmin list shadows ^| findstr GLOBALROOT') do for /f "tokens=4 delims=\" %%g in ("%%f") do mklink /d %SYSTEMDRIVE%\%%g %%f*, which may be added to a batch file to "mount all Restore Points simultaneously."[25]

6. Similarly, for the previously mentioned command *net share testshadow=\\.\HarddiskVolumeShadowCopy[volume number]*, one may automate the methodology by incorporating a *for* loop to control multiple iterations of the process such as:

`for /l %i in (start,1,stop) do *net share testshadow%i=\\.\HarddiskVolume ShadowCopy%i\*`

This method uses a *for* loop to iteratively execute the aforementioned *net share* command in order to create a share for each of the VSCs previously referenced in the *vssadmin* command and provided in the loop via the *start* and *stop* options. After one or more VSC(s) is mounted and accessible via the share(s), one may selectively view the contents using the Windows command line commands *cd* and *dir*.

7. After analysis of VSC contents is complete, one may perform "cleanup" by running the *net share {share} /DELETE* command below or by reverting to a snapshot in the surrogate, or "analysis VM."

a. `for /l %i in (start,1,stop) do net share testshadow%i /DELETE`

This uses a *for* loop to iteratively execute the *net share /DELETE* command (removes (deletes) a network share) in order to remove the network share for each of the VSCs. After executing the command, the VSCs are unmounted and are no longer accessible via the network shares.

Figures 56 through 59 depict these methodologies.

```
C:\>for /l %i in (9,1,20) do net share testshadow%i=\\.\HarddiskVolumeShadowCopy%i\

C:\>net share testshadow9=\\.\HarddiskVolumeShadowCopy9\
testshadow9 was shared successfully.

...

C:\>net share testshadow20=\\.\HarddiskVolumeShadowCopy20\
testshadow20 was shared successfully.
```

Figure 56: Executing the *net share* command

```
C:\Windows\system32>net share                (Before)

Share name   Resource                  Remark
-------------------------------------------------------------------------------
C$           C:\                       Default share
E$           E:\                       Default share
F$           F:\                       Default share
IPC$                                   Remote IPC
ADMIN$       C:\Windows                Remote Admin
The command completed successfully.

C:\Windows\system32>net share                (After)

Share name   Resource                  Remark
-------------------------------------------------------------------------------
C$           C:\                       Default share
E$           E:\                       Default share
F$           F:\                       Default share
IPC$                                   Remote IPC
ADMIN$       C:\Windows                Remote Admin
testshadow10 \\.\HarddiskVolumeShadowCopy...
...
testshadow20 \\.\HarddiskVolumeShadowCopy...
testshadow9  \\.\HarddiskVolumeShadowCopy9\
The command completed successfully.
```

Figure 58: Windows shares before and after the VSCs are mounted as "testshadowX"

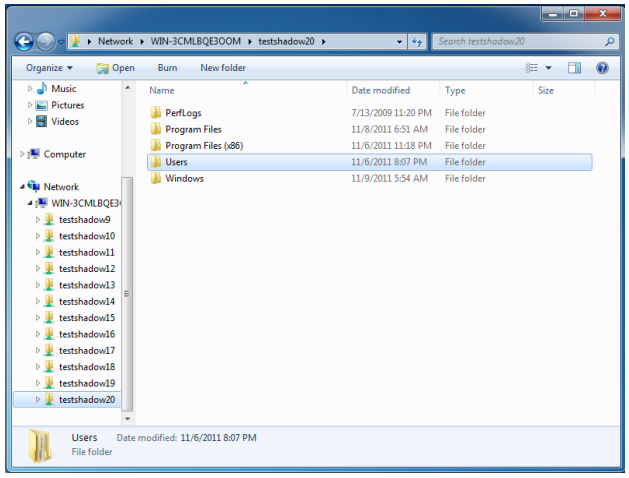


Figure 57: Mounted VSCs, now accessible via Windows shares

```
C:\>for /l %i in (9,1,20) do net share testshadow%i /DELETE

C:\>net share testshadow9 /DELETE
testshadow9 was deleted successfully.

...

C:\>net share testshadow20 /DELETE
testshadow20 was deleted successfully.
```

Figure 59: Methodology for removing the Windows shares

After providing a very brief description of the use case for the scripting manual tools method, the *robocopy* method is addressed, next.

C.2. Using *robocopy*

Robocopy has received significant interest/use from the digital investigations community, based on documented approaches for extracting VSC data by Larsen [7], Butler [31], and others. The use case for the *robocopy* method is as follows:

1. Crabtree [9] and "DC1743" [21] provide the following extraction examples using *robocopy*:

   a. `robocopy [source volume]:\[source folder] [destination volume]:\[destination folder] *.exe /S /COPY:DAT /XJ /w:0 /r:0`

   This targets all *.exe* files and recurses subfolders. */copy:dat* is the default argument defining what to copy for each file and ensures all data, attributes, and timestamps, respectively, are copied. */xj* excludes any junction points – both for directories and for files. *W:0* and *r:0*, respectively, define waiting zero seconds between retries and retrying zero times after a failed copy.

   b. `for /l %i in (2,1,3) do robocopy c:\rp%i\Users\%user% z:\Shadows\rp%i\Users\%user% *.jpg *.bmp *.png /S /COPY:DAT /XJ /w:0 /r:0`

   The *robocopy* methodology listed above uses a *for* loop to iteratively execute the aforementioned *robocopy* command (with the arguments listed) in order to copy .jpg, .bmp, and .png files for users for VSCs two and three.

2. Another methodology described in Larsen [7] and "DC1743" [21] utilizes a network share-mounted VSC as the source and copies all files (*.* is the default file type, when not specified) from subdirectories that are not empty. This method outputs its status to a specified log file, D:\VSStestcopylog.txt, and is executed as follows: `robocopy /S /R:1 /W:1 /LOG:D:\VSStestcopylog.txt \\[computername] \testshadow D:\vssTest`

Robocopy also supports the following additional arguments, which may support this research further:

/E - Copy subdirectories, including those that are empty
/NJS - Do not produce a job summary in the log file
/NHS - Do not produce a job header in the log file
/L - Produces a log only; this option disables the copying, timestamping, and/or deletion of source files
/X - Produces a report of all unselected, or "extra," files
/V - Produces verbose output (shows files skipped during the process)
/TS - Produces source file timestamps in the output
/FP - Produces the full path name of files in the output
/BYTES - Prints file sizes as bytes
/TEE - Output to the console window in addition to the log file (allows for quicker verification of results during validation testing)
/DCOPY:T - Copy directory timestamps
/CREATE - Create directory tree and zero-length files only
/Copy:copyflag[s]
- A=Attributes, T=Timestamps, S=Security=NTFS ACLs,
O=Owner info, U=aUditing info

After providing a very brief description of the use case for the *robocopy* method, the *LogParser* method is addressed, next.

C.3. Using *LogParser*

The use case for the *LogParser* method is as follows:

1. First, execute the command: `vssadmin list shadows /for=[Volume]: > "[Dest_Volume]:\VSC_Exam\VSCs.txt"` This command sends the output of the *vssadmin* command to a text file on the analysis system.

2. Mount all VSCs using the *mklink* command (as previously discussed in other sections).

3. With the VSCs mounted via symbolic directory links, in order to determine the contents of each VSC without browsing through the folders, utilize

the *LogParser* utility. *LogParser* allows one to grab the metadata for all the files within each VSC and export the metadata to CSV format. The *LogParser* command syntax to accomplish this is: *logparser -i:FS -o:CSV -preserveLastAccTime:ON "Select HASHMD5_FILE(Path),CreationTime,LastWriteTime,LastAccessTime,Name,Path,Size into '[Dest Volume]:\[Dest Path]\File.csv' From '[Source Volume]:\[Source VSC Path]*.*'"*

The *-i:FS* argument, in this instance specifying a file system, is the input type for LogParser, while the *-o:CSV* argument, in this instance specifying comma separate value, is the output type for LogParser. Edwards recommends using the "-preserveLastAccTime:ON" argument to maintain original timestamps and notes the benefit of this methodology as "a directory structure … - complete with dates and times, names and sizes of files, and MD5 hashes."[34]

LogParser also supports the following additional arguments, which may support this research further:

-useLocalTime	- Provides the option to turn off the default of using local time for dates (thus using UTC time format)
-i:FS Attributes	- Provides file attributes
-o:CSV -fileMode:0	- Causes LogParser to append to the output file if it already exists; this will be especially useful when processing multiple VSC's.
-o:SQL	- Provides for a variety of options when exporting results to SQL-formatted output

Validation testing was completed using the following command syntax: `LogParser.exe" -i:FS -o:CSV -preserveLastAccTime:ON -useLocalTime:OFF "Select

HASHMD5_FILE(Path),CreationTime,LastWriteTime,LastAccessTime,

Attributes,Name,Path,Size into D:\OutputFile.csv FROM C:\VSC17*.*"`.

After providing a very brief description of the use case for the *LogParser* method, the *Shadow Analyser* utility is briefly described, next.

C.4. *Shadow Analyser* Utility

Shadow Analyser is a Disklabs-branded utility created by Lee Whitfield and Mark McKinnon for achieving VSC data recovery.[52] McKinnon explained that the *Shadow Analyser* code was continuously updated in several areas in response to ongoing VSC structure discoveries and issues. Per online documentation, it appears the code was released to select experts for testing purposes; however, it was not found in a release that appeared available to the general public.

In the absence of a generally-released utility, sites.google.com maintains *Shadow Analyser* screenshots. Figure 60 depicts the Shadow Analyser GUI.

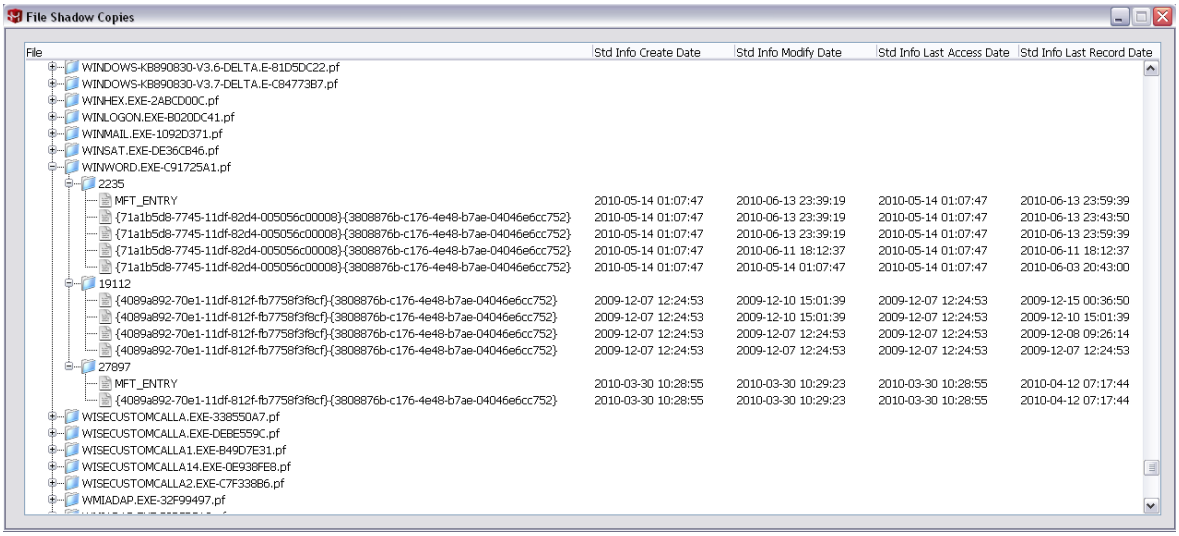


Figure 60: *Shadow Analyser* GUI as depicted on sites.google.com

This utility was not tested due to the absence of a downloadable product.

XII. Appendix D

The following table provides a summary listing of the merits and limitations, as discovered by validation testing of each of the methodologies discussed in Sections III and IV:

#	Item Description	Merit or Limitation	Reference
1	VSC must be accessed via surrogate (or native) Windows-based system that supports the Windows Previous Versions UI	Limitation	Windows Previous Versions
2	Automation of the methodology in its current form is not possible	Limitation	Windows Previous Versions
3	Required additional tools/techniques to recover files/folders from VSC(s)	Limitation	vssadmin with mklink or net share, restoring and accessing, iterative loops/scripting
4	Required additional automation to access multiple VSCs and then select and recover all metadata/data from all VSCs	Limitation	vssadmin with mklink or net share, restoring and accessing, fls and mactime, iterative loops/scripting, LogParser, ShadowCopy, ShadowExplorer, ProDiscover
5	Required additional tools/techniques to store the metadata/data in a format that allows it to be used for visualization purposes (i.e., SQL or similar format)	Limitation	vssadmin with mklink or net share, restoring and accessing, specialized utilities/methods, iterative loops/scripting, Robocopy, ShadowCopy, ShadowExplorer, ProDiscover
6	Combined date and time fields within the	Limitation	fls and mactime

	Date column		
7	Methodology resulted in parsing errors, which may require additional permissions/tools/techniques to retrieve all data	Limitation	LogParser, ShadowCopy
8	Did not extract timestamp or attribute information into the report file	Limitation	Robocopy, ShadowCopy,
9	Interface only refreshes once, at program startup time	Limitation	ShadowExplorer
10	Failed to recognize VSCs from all sources (i.e., for ProDiscover, from a .vhd file, and for ShadowExplorer, from other than the native drive upon which the running OS resided (e.g., ShadowExplorer did not allow viewing of the VSCs that were mounted via the diskpart utility))	Limitation	ShadowExplorer, ProDiscover
11	Product has a retail cost for licensing, causing the functionality to be restricted to those who may afford the license costs	Limitation	ProDiscover
12	Employed a native Windows UI for accurate, easy, and timely data recovery	Merit	Windows Previous Versions
13	Reliably accessed/mounted one or more VSCs (no metadata/data extraction performed)	Merit	vssadmin with mklink or net share With extraction -> iterative loops/scripting, ShadowCopy, ShadowExplorer, ProDiscover
14	Must be combined with another technique to extract one or more files/folders in an automated fashion as well as to maintain the original date and time stamps for restored files/folders	Merit	vssadmin with mklink or net share, restoring and accessing
15	Reliably captured and then recovered either	Merit	fls and mactime,

	metadata or data from one or more VSCs		ProDiscover
16	Quickly recursively extracted file and directory names in an automated fashion as well as sorted and formatted the metadata based on time stamp information	Merit	fls and mactime
17	May be scripted/automated for any arbitrary number of VSCs	Merit	fls and mactime, specialized utilities/methods
18	Offered flexibility of executing this methodology from an Incident Responder's toolkit in addition to other common methods -- a plus in exigent or first responder's circumstances	Merit	fls and mactime, ShadowCopy
19	Provided limited automation while maintaining the ability to keep original date and time stamps for restored files/folders	Merit	iterative loops/scripting combined with other utilities
20	Scripting enhancement may provide initial steps for enhanced automation	Merit	iterative loops/scripting
21	Extracted one or more files/folders in an automated fashion as well as maintained the original date and time stamps for restored files/folders	Merit	Robocopy, ProDiscover
22	Extracted desired metadata of files/folders in a semi-automated fashion and maintained the original date and time stamps for source files	Merit	LogParser
23	Offers complete flexibility of executing multiple third party utilities as well as with many execution argument options	Merit	specialized utilities/methods, ShadowCopy
24	Data deduplication based on file hash and handled duplicate filenames with customizable filename delimiter	Merit	ShadowCopy
25	Distinguished between processing the VSCs of the local (host) system or all non-local (external) VSCs	Merit	ShadowCopy

26	Easy-to-use graphical user interface	Merit	ShadowExplorer, ProDiscover

Table 8: Candidate Technology Merits and Limitations

XIII. Appendix E

Enhanced Shadowcopy.py code:

```
# This program browses and extracts data from VHDs that contain shadow volumes
# c :\vhd − where the VHD gets mounted
# c :\vhd\

VHD_DIR='c:\\vhd'
TESTING_ON_or_OFF=0                    # Set to 1 for testing purposes; Set to 0 for non-testing purposes
TESTING_THRESHOLD=7500                 # Establishes the number of file records to process prior to exiting

import sys,os,glob,platform,ctypes,re,sqlite3
from subprocess import call,Popen,PIPE
import ShadowVolume2


def get_vhd(fname):
  """Return the filename of the VHD for fname."""
  (root,ext) = os.path.splitext(fname)
  if ext.lower()==".vhd":
    return fname                       # it's already a vhd
                                       # See if the .vhd is there; if it isn't , make it
  vhdname = root + ".vhd"
  if os.path.exists(vhdname):
    return vhdname                     # there was a vhd there already; use it
  print ("Converting %s to %s" % (fname,vhdname))
  p = call(['vhdtool.exe','/convert',fname])
  if p!=0:
    print("Cannot convert; vhdtool returns error code %d" % p)
    exit (1)
  return vhdname


def make_needed_dirs(fn):
  "Make all of the directories required to get to path fn "
  import os.path
  if len(fn)>0 and not os.path.exists(fn):
    (head,tail) = os.path.split(fn)
    make_needed_dirs(head)
    os.mkdir(fn)


def make_filename_distinct(fn):
  """ If filename.ext exists, replace with filename.NNN.ext, where NNN is between 0 and
  if we have more than 999 files, just keep incrementing..."""
  import os.path
  if not os.path.exists(fn):
    return fn
  counter = 0
  while True:
    (path,ext) = os.path.splitext(fn)
    newfn = path+".{:03}".format(counter)+ext
    if not os.path.exists(newfn):
      return newfn


def include_volume(v,include_local):
  """Returns True if volume v should be included."""
  import platform
  if not include_local: include_local=False          # handle case of include_local==N
  return include_local == (platform.node()==v.originatingMachine()) # If platform.node name is same as VSC's originatingMachine,
                                       # return True for a printed "+" and subsequent processing

def deleteTable():                     # Drop the table ShadowCopy within the options.reportfn database
  queryCurs.execute('"DROP TABLE ShadowCopy"')


def createTable():                     # Create the table ShadowCopy within the options.reportfn database
  queryCurs.execute('"CREATE TABLE ShadowCopy
  (id INTEGER PRIMARY KEY,Path TEXT,MD5 TEXT,Size INTEGER,Machine TEXT,Volume TEXT,M_Time TEXT,A_Time TEXT,C_Time
TEXT,
  Attributes TEXT, Filename TEXT)"')
```

```
def addRecord(Path,MD5,Size,Machine,Volume,M_Time,A_Time,C_Time,Attributes,Filename): # Add a record into ShadowCopy within the
options.reportfn database
                queryCurs.execute('''INSERT INTO ShadowCopy (Path,MD5,Size,Machine,Volume,M_Time,A_Time,C_Time,Attributes,Filename)
                VALUES (?,?,?,?,?,?,?,?,?,?)''',(Path,MD5,Size,Machine,Volume,M_Time,A_Time,C_Time,Attributes,Filename))


READSIZE=65536          # read in 64kb
def process(seen,destdir,v,report):
  """Scan through the shadow volume denoted by v. Look for files
  that have a hash not in seen. Write the files not seen to dest.
  Save results in report.
  """
  import mmap,hashlib,csv,datetime
  global options
# Testing only, next line ->
  testnum=0                             # Initialize testing counter variable, testnum, to zero
# Process directory
  for (dirpath,dirnames,filenames) in os.walk(v.volumePath()):
    try:
      st = os.stat(dirpath)                 # Run the stat command against the directory
      original_dn = dirpath.replace(v.volumePath(),"");  # Remove the VSC path & save as orig dirname
# Call attrib and then addRecord here to add the dirpath to the database
# Start the gather attribute information section while processing directories
      p = os.popen('attrib ' + dirpath)  # Calls the Windows attrib.exe binary; may want to include another option for non-Windows platforms
      t = p.read()                  # Perform read to get the results from the attrib call
      if t[:12]=="Parameter fo" or "Path not fou":       # Check first 12 chars of attributes; if "Parameter fo" or "Path not fou,"
        t=""                        # then discard and save an empty result (command could not retrieve attributes).
      p.close()                     # Cleanup
# End the gather attribute information section
# Start the addRecord section while processing directories
      addRecord(original_dn,            #Record the dirpath
            "0 (dirpath)",          #Record "0 (dirpath)" instead of MD5 hash
            os.path.getsize(dirpath),
            v.originatingMachine(),
            v.volumeName(),
            datetime.datetime.fromtimestamp(st.st_mtime),
            datetime.datetime.fromtimestamp(st.st_atime),
            datetime.datetime.fromtimestamp(st.st_ctime),
            t[:12],                 # Save attribute info, which is the first 12 chars of output of the attrib command
            original_dn)
# End the addRecord section for directories
    except (WindowsError) as ex:
      print("Windows cannot read: {}; \n{} continuing ... ".format(dirpath, str(ex)),file=error_report) # Log the exception
      continue
    except (IOError) as ex:
      print("Windows cannot open: {}; \n{} continuing ... ".format(dirpath, str(ex)),file=error_report) # Log the exception
      continue


# Process files
    for filename in filenames:              # For each file in the array of filenames, do...
# Testing segment code -- exits gracefully when TESTING_THRESHOLD records have been processed
      if TESTING_ON_or_OFF==1:              # If the testing flag is turned on ...
        testnum=testnum+1                   # Increase testnum by one
        if testnum==TESTING_THRESHOLD:      # If TESTING_THRESHOLD records are in the DB, then ...
          report.commit()                   # Commit the changes to the database and
          queryCurs.close()                 # Close the cursor to the database file and
          exit(0)                           # Exit gracefully.
# End of testing segment code
      shadow_fn = os.path.join(dirpath,filename)# Join the directory path and filename to create a complete path
      try:
        st = os.stat(shadow_fn)             # Run the stat command against the file
        if options.minsize <= st.st_size <= options.maxsize:    # If the files size is within range, then...
          with open(shadow_fn,"rb") as f:       # Open the path as a file in readonly, binary mode
            map = mmap.mmap(f.fileno(),length=0,access=mmap.ACCESS_READ)  # assign to map
            md5 = hashlib.md5(map)          # MD5Sum the file
            original_fn = shadow_fn.replace(v.volumePath(),"");   # Remove the VSC path & save as orig filename
# Start the gather attribute information section while processing files
            p = os.popen('attrib ' + shadow_fn) # Calls the Windows attrib.exe binary file; may want to include another option for non-Windows platforms
            t = p.read()                    # Perform read to get the results from the attrib call
            if t[:12]=="Parameter fo" or "Path not fou":        # Check first 12 chars of attributes; if "Parameter fo" or "Path not fou" ...
              t=""                          # then discard and save an empty result (command could not retrieve attributes).
            p.close()                       # Cleanup
# End the gather attribute information section
# Start the addRecord section while processing files
            addRecord(original_fn,
                  md5.hexdigest(),
                  os.path.getsize(shadow_fn),
                  v.originatingMachine(),
                  v.volumeName(),
```

```
                datetime.datetime.fromtimestamp(st.st_mtime),
                datetime.datetime.fromtimestamp(st.st_atime),
                datetime.datetime.fromtimestamp(st.st_ctime),
                t[:12],                 # Save attribute info, which is the first 12 chars of output of the attrib command
                filename)
# End the addRecord section for files
# If we are extracting, make the directories and copy the data
        if not options.noextract:           # If extracting...
          if md5.digest() not in seen:      # If the file is not in seen array, then...
                                # This file hasn't been seen before.
                                # Write the file's info to the report and copy it to the dest
            dest_fn = shadow_fn.replace(v.volumePath(),destdir)   # Replace destdir for VSC path & save as dest_fn
            dest_fn = make_filename_distinct(dest_fn)     # Ensure filename.NNN.ext if dup filename
          make_needed_dirs(os.path.dirname(dest_fn))
          # Now copy over the file data
          map.seek(0)
          with open(dest_fn,"wb") as fdest:
            while True:
              buf = map.read(READSIZE)
              if len(buf)==0:                   # End of file!
                break
              fdest.write(buf)
          # put back times
          os.utime(shadow_fn,(st.st_atime,st.st_mtime))
          os.utime(dest_fn,(st.st_atime,st.st_mtime))
        # Now we've seen this file!
        seen.add(md5.digest())              # Add the md5sum for the file just hashed into the seen[] hash array
    except (WindowsError) as ex:
      print("Windows cannot read: {}; \n{} continuing ... ".format(shadow_fn, str(ex)),file=error_report) # Log the exception
      continue
    except (IOError) as ex:
      print("Windows cannot open: {}; \n{} continuing ... ".format(shadow_fn, str(ex)),file=error_report) # Log the exception
      continue
  report.commit()                           # Commit the changes to the database


if __name__=="__main__":
  from optparse import OptionParser
  global options
  import sys,time,datetime

  parser = OptionParser()
  parser.usage = """usage: %prog [options] <EXTRACT-DIR>

<imagefile> may be a .vhd or a .raw.  If it is a .raw, it will
be converted to a .vhd IN PLACE, so be sure you have enough disk
and the vhdtool.exe to do the conversion

Note: this script must be run as administrator.
"""
# *****Note: The --image option was removed and the --mount and --unmount options were added*****
  parser.add_option("--mount",help="Prompts the user for a vdisk (VHD).  Then, mounts the selected image.",
                action="store_true")
  parser.add_option("--list",help="Show the shadow volumes that are available.",
                action="store_true")
  parser.add_option("--local",help="Analyze only the local machine",
                action="store_true")
  parser.add_option("--maxsize",help="Specifies maximum size of a file to extract",
                type='int',default=1024*1024*1024*1024)
  parser.add_option("--minsize",help="Specifies minimum size of a file to extract",
                type='int',default=1)
  parser.add_option("--noextract",help="Do not extract the shadow data",
                action="store_true")
  parser.add_option("--reportfn",help="Specify report output filename (default='report.Db')",
                default="report.Db")
  parser.add_option("--zap",help="Overwrite report file if it exists",
                action="store_true")
  parser.add_option("--unmount",help="Unmount a selected VHD image",
                action="store_true")
  if len(sys.argv)==1:
     parser.print_help()
     exit(0)

  global options
  (options,args) = parser.parse_args()


  if not ctypes.windll.shell32.IsUserAnAdmin():
     os.system("color c")
     print("")
```

```
    print("***** This script must run as the Windows Administrator. *****\n")
    time.sleep(1)
    os.system("color 07")
    print("")
    exit (1)

# Gather the VHD image path so that we can mount/unmount the vdisk
# Then, call diskpart_mount() here with the script to mount the vdisk
if (options.mount):                         # if the --mount option was selected, then...
    imagep = ShadowVolume2.image()          # prompt the user for the VHD path and confirm it
    time.sleep(2)                           # sleep for two seconds
    ShadowVolume2.diskpart_mount(imagep)    # mount the image
    os.system("color 2")
    print("\nPlease enter next command:\n")
    time.sleep(3)                           # Sleep for three seconds since diskpart execution requires time.
    parser.print_help()
    os.system("color 07")
    print("")
    exit(0)

# Call diskpart_unmount() here with the script to unmount the vdisk
if (options.unmount):                       # if the --unmount option was selected, then...
    ShadowVolume2.diskpart_unmount()        # unmount the image using the existing diskpart script from the mount option
    time.sleep(3)                           # sleep for three seconds
    exit(0)

# Get all volumes (local and non—local) for the --list listing
vols = ShadowVolume2.availableVolumes()     # Call ShadowVolume2 to get all VSCs
if (options.list):                          # if the --list option was selected, then...
    include_legend = {True:"+",False:" "}   # Set True="+" and False=SPACE
    fmt ="{:1} {:15} {:25} {}"              # Format the output columns
    print(fmt.format("","Source", "Creation Time", "Volume Name"))  # Format column headers
    print(fmt.format("","------", "-------------", "-----------"))  # Format column spacers
    for v in vols:                                  # For all VSCs....
        print(fmt.format(include_legend[include_volume(v,options.local)],   # Print either a "+" or a SPACE
                v.originatingMachine(),v.ctime(),v.volumeName()))  # Print source system, CTime, VSC name
    print("")
    print("+ means volume will be included in analysis")
    exit(0)

seen = set()                            # for seen MD5 hashes, call set().  ???This loads the seen[] array.???
destdir = ""
if not options.noextract:                   # if the --noextract option was NOT selected, then...
    if len(args)!=1:                        # if there is not an argument for the extraction directory, then...
        os.system("color c")
        print("")                          # Demand that we did not get an extract dir
        print("***** No extraction directory provided. *****\n")
        parser.print_help()
        time.sleep(1)
        os.system("color 07")
        print("")
        exit (1)
    destdir = args[0]


if os.path.exists(options.reportfn) and not options.zap: # if report file exists and --zap was NOT selected, then...
    os.system("color c")
    print("{} exists. Delete it via the --zap option or specify a new report filename with --report option.\n".format(options.reportfn))
                                # Print error message to stdout...
    time.sleep(1)
    os.system("color 07")
    print("")
    exit (1)

if os.path.exists(options.reportfn) and options.zap:    # if report file exists and --zap WAS selected, then...
    os.remove(options.reportfn)
    os.system("color 2")                # Provide user feedback message to stdout...
    print("\n{} existed; however, was removed per the user-specified --zap option.\n\n".format(options.reportfn))
    time.sleep(2)
    os.system("color 07")
    print("")

# Timekeeping code for determining processing time
current_time = datetime.datetime.now()
current_time_text = current_time.strftime("%Y-%m-%d %H:%M:%S")
print("\n\n\nStart time is: {}".format(current_time_text))

error_report = open("error_log.txt",'w',encoding='utf —8')  # Open the error report filename in write mode
report = sqlite3.connect(options.reportfn)    # Create a new Db file or opens existing Db file and provide a handle
queryCurs = report.cursor()                 # Set a cursor to allow Python to iterate through the database file
```

```
  createTable()                         # Create the table ShadowCopy within the options.reportfn database.
                                        # The ShadowCopy table in the report database is an array of arrays of metadata.

  for v in vols:                        # For the current to last VSC...
     if not include_volume(v,options.local):   # If current VSC should be "included" (i.e., had a + mark), then...
        continue
     os.system("color 2")
     print(" Processing {} from {}".format(v.volumeName(),v.originatingMachine())) # Alert user which VSC is getting processed now
     process(seen,destdir,v,report)          # Walk the VSC, processing all folders and files...
     report.commit()                    # Commit the changes to the database
     os.system("color 07")
     print("")
     time.sleep(2)

#  deleteTable()                        # Delete the table ShadowCopy so that it will not exist for subsequent testing purposes
  queryCurs.close()                      # Close the cursor to the database file
# End of timekeeping code for determining processing time
  current_time = datetime.datetime.now()
  current_time_text = current_time.strftime("%Y-%m-%d %H:%M:%S")
  print("\n\n\nEnd time is: {}\n\n".format(current_time_text))
```

Enhanced ShadowVolume2.py code:

```
# Code taken from Brian Madden
import re,os,time
from subprocess import call,Popen,PIPE

class ShadowCopy:
  def __init__(self, attrs):
    self.attrs = attrs
  def originatingMachine(self):
    return self.attrs['Originating Machine']
  def ctime(self):
    return self.attrs['creation time']
  def volumePath(self):
    return self.attrs['Shadow Copy Volume']
  def volumeName(self):
    return self.volumePath().split('\\')[-1]

def vssadmin_list_parse(vssadmin_out):
  """ This function takes the output from the vssadmin command and returns
a set of objects that describe each shadow copy"""
  ret = []       # list to return
  current = None              # copy we are currently processing
  fix1 = re.compile("Contained.*copies at ")  # Find the phrase "Contained.....copies at "
  for line in vssadmin_out.splitlines():
    if line=="":
      if current:
        ret.append(ShadowCopy(current))
        current = None
        continue
    if line.startswith("Contents of shadow copy set ID:"):  # Find the phrase ...
      current = {}              # start of the new one
      id = line.split(":")[1][1:]
      current['id'] = id  # Set the current[id] to the VSC GUID we're looking at
      continue
    if not current:             # not in the data
      continue
    # If we get here, we have a name, a colon, and a value
    line = line.strip()         # remove whitepsace
    colon = line.find(':') # we only want to work with the first colon, so we can't
    if colon >=0:
      name = line[0:colon]
      value = line[colon+2:]  # Set value equal to 2 columns after the colon (i.e., the creation date/time)
      if name.endswith("creation time"): name="creation time" # if there is no creation time, set it to "creation time" to avoid NULL
      current[name] = value   # Set the current name
  return ret

def availableVolumes():
  " Return a list of available shadow volumes"
  list_output = Popen(['vssadmin.exe','list','shadows'], stdout=PIPE).communicate()[0]
  list_output = list_output.decode('utf -8')
  return vssadmin_list_parse(list_output)

def diskpart_mount(mountpath):
  """Run diskpart.exe with the mount script and print results"""
  dpscript = open('diskpart_script1.txt','w',encoding='utf -8')       # Open the diskpart script in write mode
```

```
  print('SELECT','VDISK','FILE=',mountpath,file=dpscript)            # Print the select statement into the diskpart script
  print('ATTACH','VDISK','READONLY',file=dpscript)                   # Print the attach statement into the diskpart script
  dpscript.close()                                                   # Close the diskpart script file
  mountvar = Popen(['diskpart','/s','diskpart_script1.txt'], stdout=PIPE).communicate()[0]  # Call diskpart with the /s option to run the script
  print(mountvar.decode('utf -8'))                                   # Output results

def diskpart_unmount():
  """Run diskpart.exe with the unmount script and print results"""
  if os.path.exists('diskpart_script1.txt'):                         # If the diskpart script file exists...
    dpscript = open('diskpart_script1.txt','r')                      # Open the diskpart script in read mode
    lines = dpscript.readlines()                                     # Count the number of lines in the file
    dpscript.close()                                                 # Close the diskpart script file
    w = open('unmount_diskpart_script1.txt','w',encoding='utf -8')   # Open the unmount diskpart script file in write mode
    w.writelines([item for item in lines[:-1]])                      # Write all lines but the last (all but the attach statement)
    print('DETACH','VDISK',file=w)                                   # Print the detach statement into the diskpart script
    w.close()                                                        # Close the unmount diskpart script file
    mountvar = Popen(['diskpart','/s','unmount_diskpart_script1.txt'], stdout=PIPE).communicate()[0]  # Call diskpart with the /s option to run the script
    print(mountvar.decode('utf -8'))                                 # Output results
    time.sleep(2)                                                    # Sleep two seconds prior to removing files since diskpart takes time
    os.remove('diskpart_script1.txt')                                # Since unmount was successful, remove the diskpart_script1.txt file
    os.remove('unmount_diskpart_script1.txt')                        # Since unmount was successful, remove the unmount_diskpart_script1.txt file
  else:
    print("Cannot find record of mounted image file (i.e., diskpart_script1.txt file).") # If diskpart script is not found, print an error message

def image():
  """Prompt the user for a vdisk (VHD) image path to mount"""
  response="N"                                                       #set response to '(n)o'
  while response not in ("y","Y"):                                   #enter loop
      imagepath = input("\nPlease enter the path of the vdisk (VHD) file to mount. -->")  #prompt the user for the path and record it to variable imagepath
      print("You entered: "+imagepath+"\n")                          #provide user with feedback on the path he/she provided
      response = input("Is this correct? (Enter 'Y' or 'y') -->") #require user confirmation on the validity of the path
      if os.path.exists(imagepath) and response in ("y","Y"): #if the path is legitimate and the user confirmed '(y)es', then
        break                                                        #exit loop
      else:                                                          #else
        if response in ("y","Y"):                                    #if the user confirmed '(y)es' and the path is not valid
          print("The path of the vdisk (VHD) file to mount, "+imagepath+", was not found.\n") #let the user know the path is invalid
        response="N"                                                 #set the response to '(n)o' since the path is invalid
  print("\nThe vdisk (VHD) path has been saved as: "+imagepath+"\n")  #after getting out of the loop with a valid path and user confirmation, print this
msg
  return imagepath                                                   #return the path to the image to process
```

XIV. Glossary

Quiesce: *Quiesce* is used to describe pausing or altering the state of running processes on a computer, particularly those that might modify information stored on disk during a backup, in order to guarantee a consistent and usable backup. This generally requires flushing any outstanding writes. See also: buffering.[53]

Virtual Hard Drive: When an entire system is backed-up, a *Virtual Hard Drive* (VHD) file is created. This file may be mounted as a virtual disk. "The VHD format captures the entire virtual machine operating system and the application stack in a single file."[38]

XV. References/Bibliography

1. time line. Dictionary.com. Dictionary.com Unabridged. Random House, Inc. http://dictionary.reference.com/browse/time%20line (accessed: October 02, 2011).
2. Volume Shadow Copy Service Overview. Microsoft.com. http://msdn.microsoft.com/en-us/library/aa384649(v=vs.85).aspx (accessed: October 02, 2011).
3. Volume Shadow Copy Service. Microsoft.com. http://msdn.microsoft.com/en-us/library/bb968832(v=vs.85).aspx (accessed: October 02, 2011).
4. Leschke, T. R., Rheingans, P., and Sherman, A. T. Using a Fisheye View to Visualize Change-Over-Time in Support of Digital Forensic Examinations. Department of Computer Science, University of Maryland Baltimore County, August 2011.
5. Daniel L. and Daniel L. Digital Forensics for Legal Professionals – Understanding Digital Evidence from the Warrant to the Courtroom. Elsevier Inc. 2012. ISBN 978-1-59749-643-8. 207-211. Available from http://www.sciencedirect.com/science/book/9781597496438#ancs4 (accessed: November 13, 2011).
6. MAC times. Wikipedia.org. http://en.wikipedia.org/wiki/MAC_times (accessed: October 22, 2012).
7. Windows 7: Current Events in the World of Windows Forensics. SANS.org. http://computer-forensics.sans.org/summit-archives/2010/files/12-larson-windows7-foreniscs.pdf (accessed: October 02, 2011).
8. Russinovich, M. E., Solomon, D. A., and Ionescu A. Windows Internals Part 2. Microsoft Press, Redmond, 2012.
9. Reliably recovering evidential data from Volume Shadow Copies in Windows Vista and Windows 7. QCCIS.com. http://www.qccis.com/downloads/whitepapers/QCC%20VSS%20Whitepaper.pdf (accessed: October 02, 2011).
10. Windows Vista "Time Warp": Understanding Vista's Backup and Restore Technologies. MSDN.com. http://channel9.msdn.com/posts/Charles/Windows-Vista-quotTime-Warpquot-Understanding-Vistas-Backup-and-Restore-Technologies/ (accessed: December 03, 2011).
11. Shadow copies - a peek under the hood. MSDN.com. http://blogs.msdn.com/b/adioltean/archive/2004/12/11/280052.aspx (accessed: October 23, 2012).
12. What you should know about Volume Shadow Copy/System Restore in Windows 7 & Vista (FAQ). Szynalski.com. http://blog.szynalski.com/2009/11/23/volume-shadow-copy-system-restore/ (accessed: October 23, 2012).
13. File vs Block Level Backups. Inetu.net. http://blog.inetu.net/2011/03/file-vs-block-level-backups/ (accessed: December 8, 2011).
14. Shadow Warriors. SANS.org. http://computer-forensics.sans.org/summit-archives/2010/files/36-whitefield-shadow-warriors.pdf (accessed: September 12, 2011).
15. Libvshadow. Google.com. http://code.google.com/p/libvshadow/ (accessed: October 22, 2012).
16. Into The Shadows. Forensic4cast.com. http://www.forensic4cast.com/2010/04/into-the-shadows (accessed: October 13, 2011).

17. CreateFile function. Microsoft.com. http://msdn.microsoft.com/en-us/library/windows/desktop/aa363858(v=vs.85).aspx (accessed: October 15, 2012).
18. Shadow Timelines and Other VolumeShadowCopy Digital Forensics Techniques with the Sleuthkit on Windows. SANS.org. http://computer-forensics.sans.org/blog/2010/03/16/shadow-timelines-and-other-shadowvolumecopy-digital-forensics-techniques-with-the-sleuthkit-on-windows/ (accessed: September 11, 2011).
19. VISTA and Windows 7 Shadow Volume Forensics. SANS.org. http://computer-forensics.sans.org/blog/2008/10/10/shadow-forensics (accessed: October 02, 2011).
20. Accessing Volume Shadow Copies. Blogspot.com. http://windowsir.blogspot.com/2011/01/accessing-volume-shadow-copies.html (accessed: October 02, 2011).
21. Volume Shadow Copy Forensics – the Robocopy method Part 2. Blogspot.com. http://forensicsfromthesausagefactory.blogspot.com/search?q=volume+shadow+copies (accessed: October 02, 2011).
22. A Little Help with Volume Shadow Copies. Blogspot.com. http://journeyintoir.blogspot.com/2011/04/little-help-with-volume-shadow-copies.html (accessed: October 11, 2011).
23. Vssadmin. Microsoft.com. http://technet.microsoft.com/en-us/library/cc754968(v=ws.10).aspx (accessed: January 8, 2012).
24. HowTo: Mount and Access VSCs. Blogspot.com. http://windowsir.blogspot.com/2011/09/howto-mount-and-access-vscs.html (accessed: October 11, 2011).
25. Hargreaves C, Chivers H, Titheridge D. Windows Vista and digital investigations. Digital Investigation September 2008;5(1–2): 34–48. Available from: http://www.sciencedirect.com/science/article/B7CW4-4T9CCWR-1/2/3253ca2acfa2301766db3deddb2d55e4 (accessed: November 03, 2011).
26. A simple way to access Shadow Copies in Vista. MSDN.com. http://blogs.msdn.com/adioltean/archive/2008/02/28/a-simple-way-to-access-shadow-copies-in-vista.aspx (accessed: December 3, 2011).
27. Disk image. Wikipedia.org. http://en.wikipedia.org/wiki/Disk_image (accessed: October 22, 2012).
28. Volume Shadow Copy with ProDiscover. TechPathways.com. http://toorcon.techpathways.com/uploads/VolumeShadowCopyWithProDiscover-0511.pdf (accessed: October 02, 2011).
29. VSC Parser. Mark McKinnon. Received via email. (accessed: November 11, 2011).
30. The Sleuth Kit (TSK). Sleuthkit.org. http://www.sleuthkit.org/ (accessed: April 24, 2012).
31. Timelines. Sleuthkit.org. http://wiki.sleuthkit.org/index.php?title=Timelines (accessed: October 23, 2012).
32. The Sleuth Kit Part 3 - fls, mactime and icat. Sysforensics.org. http://www.sysforensics.org/2012/03/sleuth-kit-part-3-fls-mactime-and-icat.html (accessed: October 23, 2012).
33. Robocopy a Computer Forensics tool? SANS.org. http://computer-forensics.sans.org/blog/2009/01/08/robocopy-a-computer-forensics-tool/ (accessed: October 11, 2011).

34. Volume Shadow Copies and LogParser. SANS.org. http://computer-forensics.sans.org/blog/2011/06/09/vscs-logparser (accessed: October 02, 2011).
35. Hom, M. Shadowcopy: A python-based shadow volume enumeration and digest tool. Naval Postgraduate School, September 2011.
36. ShadowExplorer. ShadowExplorer.com. http://www.shadowexplorer.com/downloads.html (accessed: October 02, 2011).
37. dd (Unix). Wikipedia.org. http://en.wikipedia.org/wiki/Dd_(Unix) (accessed: October 22, 2012).
38. Microsoft Virtual Hard Disk Overview. Microsoft.com. http://technet.microsoft.com/en-us/bb738373 (accessed: December 8, 2011).
39. VMDK. Wikipedia.org. http://en.wikipedia.org/wiki/VMDK (accessed: October 22, 2012).
40. Volume Shadow Copy Forensics – the Robocopy method Part 1. Blogspot.com. http://forensicsfromthesausagefactory.blogspot.com/2010/04/volume-shadow-copy-forensics-robocopy.html (accessed: October 02, 2011).
41. Vista Volume Shadow Copy issues. Blogspot.com. http://forensicsfromthesausagefactory.blogspot.com/2009/08/vista-volume-shadow-copy-issues.html (accessed: October 02, 2011).
42. Data deduplication. Wikipedia.org. http://en.wikipedia.org/wiki/Data_deduplication (accessed: October 22, 2012).
43. Portable Python. Portablepython.com. http://www.portablepython.com/ (accessed: October 22, 2012).
44. sqlite3 – Embedded Relational Database. DougHellmann.com. http://www.doughellmann.com/PyMOTW/sqlite3/ (accessed: September 8, 2012).
45. How Volume Shadow Copy Service Works. Microsoft.com. http://technet.microsoft.com/en-us/library/cc785914(WS.10).aspx (accessed: January 9, 2012).
46. Diskshadow. Microsoft.com. http://technet.microsoft.com/en-us/library/cc772172(v=ws.10).aspx (accessed: October 22, 2012).
47. Volume Shadow Copy Service. Microsoft.com. http://technet.microsoft.com/en-us/library/ee923636(WS.10).aspx (accessed: December 8, 2011).
48. Working with Volume Shadow Copies. Blogspot.com. http://windowsir.blogspot.com/2009/11/working-with-volume-shadow-copies.html
49. Volume Shadow Copy Forensics..cannot see the wood for the trees? Blogspot.com. http://forensicsfromthesausagefactory.blogspot.com/2010/02/volume-shadow-copy-forensics-cannot-see.html (accessed: October 02, 2011).
50. Seizing Volume Shadow Copies. ForensicFocus.com. http://www.forensicfocus.com/index.php?name=Forums&file=viewtopic&t=3428 (accessed: October 3, 2011).
51. Understanding the UserAssist Registry Key. Accessdata.com. http://accessdata.com/downloads/media/UserAssist%20Registry%20Key%209-8-08.pdf (accessed: April 24, 2012).
52. Shadow Analyser. Google.com. https://sites.google.com/a/shadowanalyser.com/shadow-analyser/home (accessed: October 02, 2011).

53. Quiesce. Wikipedia.org. http://en.wikipedia.org/wiki/Quiesce (accessed: April 30, 2012).

Other references:
54. Russinovich, Mark E., Solomon, David A., and Ionescu, Alex. Windows Internals: Covering Windows Server 2008 and Windows Vista. Microsoft Press. 2009. (pp. 688-698)
55. Exchange Server 2003 data backup and Volume Shadow Copy services. Microsoft.com. http://support.microsoft.com/kb/822896 (accessed: January 9, 2012).
56. Win32_ShadowCopy Class. Microsoft.com. http://msdn.microsoft.com/en-us/library/aa394428%28VS.85%29.aspx (accessed: November 14, 2011).
57. Leschke, Timothy R. Shadow Spyder: A Coordinated and Multiple View Tool for Browsing Shadow Volume Data, Doctoral Dissertation Proposal (Draft). Department of Computer Science, University of Maryland Baltimore County, April 2012.
58. WhatChanged for Windows 4.1.0. Brothersoft.com. http://www.brothersoft.com/whatchanged-for-windows-15357.html (accessed: October 02, 2011).
59. Process Monitor v2.96. Microsoft.com. http://technet.microsoft.com/en-us/sysinternals/bb896645 (accessed: October 02, 2011).
60. Harms K. Forensic analysis of system restore points in Microsoft windows XP. Digital Investigation September 2006; 3(3):151–8.
61. Windows Incident Response: SANS Forensic Summit. BlogSpot.com. http://windowsir.blogspot.com/2008/10/sans-forensic-summit.html (accessed: December 03, 2011).
62. What's New in VSS in Windows Vista. Microsoft.com. http://msdn.microsoft.com/en-us/library/aa819772.aspx (accessed: December 8, 2011).
63. Volume Shadow Copy Service Tools and Settings. Microsoft.com. http://technet.microsoft.com/en-us/library/cc787108(v=ws.10).aspx (accessed: January 9, 2012).
64. Volume Shadow Copy API Reference. Microsoft.com. http://msdn.microsoft.com/en-us/library/windows/desktop/aa384648(v=vs.85).aspx (accessed: January 9, 2012).
65. A brief study of time. CDN.com. http://ac.els-cdn.com/S1742287607000394/1-s2.0-S1742287607000394-main.pdf?_tid=ae717d93d7b154c83c79f0521c34d8d2&acdnat=1335145012_029ee28f963cdffcdc2e8f9757626580 (accessed: April 22, 2012).
66. Fls. Sleuthkit.org. http://www.sleuthkit.org/sleuthkit/man/fls.html (accessed: April 24, 2012).
67. Mactime. Sleuthkit.org. http://wiki.sleuthkit.org/index.php?title=Mactime (accessed: April 24, 2012).
68. How do I setup the Shadow Copy Client? How do I use the Shadow Copy feature? Petri.co.il. http://www.petri.co.il/how_to_use_the_shadow_copy_client.htm
69. EnCase Physical Disk Emulator. GuidanceSoftware.com. http://www.guidancesoftware.com/computer-forensics-software-disk-emulator.htm
70. IMDisk Virtual Disk Driver. Ltr-data.se. http://www.ltr-data.se/opencode.html/#ImDisk

71. Robocopy. Microsoft.com. http://technet.microsoft.com/en-us/library/cc733145%28WS.10%29.aspx
72. Shadow Copy. Wikipedia.org. http://en.wikipedia.org/wiki/Shadow_Copy (accessed: January 9, 2012).
73. StarWind V2V Converter. StarWindSoftware.com. http://www.starwindsoftware.com/converter
74. Run IT on a Virtual Hard Disk – Test Drive Program. Microsoft.com. http://technet.microsoft.com/en-us/bb738372.aspx
75. VHDTool. Microsoft.com. http://archive.msdn.microsoft.com/vhdtool
76. What is WinImage? WinImage.com. http://www.winimage.com/winimage.htm
77. Win32_ShadowDiffVolumeSupport Class. Microsoft.com. http://msdn.microsoft.com/en-us/library/aa394429(v=VS.85).aspx
78. VMDK to VHD Converter Available. VMToolkit.com. http://vmtoolkit.com/blogs/announcements/archive/2006/11/20/vmdk-to-vhd-converter-available.aspx (accessed: November 14, 2011).
79. DiskPart. Microsoft.com. http://technet.microsoft.com/en-us/library/cc770877%28WS.10%29.aspx (accessed: October 22, 2012).
80. What you should know about Volume Shadow Copy/System Restore in Windows 7 & Vista (FAQ). Szynalski.com. http://blog.szynalski.com/2009/11/23/volume-shadow-copy-system-restore/ (accessed: October 11, 2011).
81. Detecting data theft using stochastic forensics. DFRWS.org. http://dfrws.org/2011/proceedings/13-345.pdf (accessed: September 11, 2011).
82. Computer forensic timeline visualization tool. ScienceDirect.com. http://www.sciencedirect.com/science/article/pii/S1742287609000425 (accessed: April 22, 2012).
83. Incident Response, Forensics, & Looking for Bear Tracks. SlideShare.net. http://www.slideshare.net/ctin/msra-2011-windows7-forensicstroyla (accessed: November 13, 2011).
84. Practical Time Travel: A "Time Machine" for Windows 7. BearsOnTheLoose.com. http://www.bearsontheloose.com/images/Practical_Time_Travel_4_.pdf (accessed: October 11, 2011).
85. Digital Forensics and Windows 7 Overview. SlideShare.Net. http://www.slideshare.net/ctin/windows-7-forensics-overviewr3 (accessed: September 11, 2011).
86. Yuandong Zhu, Joshua James, Pavel Gladyshev. A comparative methodology for the reconstruction of digital events using windows restore points, Digital Investigation, Volume 6, Issues 1–2, September 2009, Pages 8-15, ISSN 1742-2876, 10.1016/j.diin.2009.02.004. (http://www.sciencedirect.com/science/article/pii/S1742287609000280).
87. FILETIME Structure. Microsoft.com. http://msdn.microsoft.com/en-us/library/windows/desktop/ms724284(v=vs.85).aspx (accessed: January 9, 2012).
88. Incremental backup vs differential backup: difference and benefits of each one. Acronis.com. http://www.acronis.com/resource/solutions/backup/2005/incremental-backups.html (accessed: October 21, 2012).