NPS-CS-11-010

NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

SHADOWCOPY: A PYTHON-BASED SHADOW VOLUME ENUMERATION AND DIGEST TOOL

by

Mike Hom

September 12, 2011

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

Daniel T. Oliver
President

Leonard A. Ferrari
Executive Vice President and
Provost

This report was prepared for and funded by the Defense Intelligence Agency, Washington, DC.

Reproduction of all or part of this report is authorized.

This report was prepared by:

Mike Hom
Department of Computer Science

Reviewed by:

Peter Denning
Chairman
Department of Computer Science

Released by:

Karl A. van Bibber, Ph.D.
Vice President and Dean of Research

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE							*Form Approved OMB No. 0704–0188*

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704–0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202–4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE *(DD–MM–YYYY)* 12–9–2011	**2. REPORT TYPE** Technical Report	**3. DATES COVERED** *(From — To)* 2011-06-26—2011-09-17
4. TITLE AND SUBTITLE shadowcopy: A python-based shadow volume enumeration and digest tool		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Mike Hom		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943		**8. PERFORMING ORGANIZATION REPORT NUMBER** NPS-CS-11-010
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Intelligence Agency		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		
13. SUPPLEMENTARY NOTES The views expressed in this report are those of the author and do not necessarily reflect the official policy or position of the Department of Defense or the U.S. Government.		

14. ABSTRACT

This report presents *shadowcopy*, tool written in Python that extracts and deduplicates files from Microsoft NTFS Shadow copies using the Microsoft Volume Shadow Service (VSS), copies the files to an external volume, and prepares a report of each extracted file's name, timestamp, original path, and MD5 hash value.

15. SUBJECT TERMS

Microsoft, Windows, Vista, 7, Shadow Volume, Volume Shadow Service

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(include area code)*
Unclassified	Unclassified	Unclassified	UU	23	

NSN 7540-01-280-5500 **Standard Form 298 (Rev. 8–98)** Prescribed by ANSI Std. Z39.18

THIS PAGE INTENTIONALLY LEFT BLANK

Abstract

This report presents *shadowcopy*, tool written in Python that extracts and deduplicates files from Microsoft NTFS Shadow copies using the Microsoft Volume Shadow Service (VSS), copies the files to an external volume, and prepares a report of each extracted file's name, timestamp, original path, and MD5 hash value.

1 Introduction

Microsoft shadow volumes are snapshots of a Windows NTFS file system that are "a read-only point-in-time replica of an original volume's contents. Each shadow copy is keyed by a persistent GUID." [7]

Forensic analysts can use shadow volumes to recover intact files that were once present in the file system but were deleted by the user or the system. Typically shadow volume snapshots take minimal space because the majority of files in the file system do not change between snapshots. This can cause additional work for the forensic examiner, however.

General-purpose tools to analyze Microsoft shadow volumes are included with Windows 7; tools specially designed for forensic examiners have been available since at least 2009. However, most of these tools are manual. and have been available to forensic has been readily available but much of it has historically been a manually driven process.

Automating the extraction of distinct files from shadow volumes saves the examiner time and is vital for additional forensic pipeline automation.

This paper describes the Microsoft Volume Shadow Copy Service (VSS), presents a brief analysis of currently available tools, and present *shadowcopy.py*, a short Python program that performs extraction, deduplication and report generation.

2 Background

2.1 Volume Shadow Copy Service (VSS)

The Volume Shadow Copy Service was first implemented in Windows XP but differed from later instantiations of VSS as it created non-persistent snapshots for file-based backups. Windows Server 2003 introduced the more commonly encountered version of VSS that creates persistent snapshots.

There are two methods for creating shadow copies. The first is to make a *complete* copy and the second is to create a *differential* copy. When the full copy (also called a *clone*) is made, the copy remains synchronized until the connection is broken for the shadow copy. At this point, the source data and shadow copy volume are independent of each other. This shadow copy volume is now a snapshot of the original volume, timestamped at the point it broke the synchronization connection and is a read-only copy to maintain integrity.

When a *differential* copy is initiated, an initial copy of the original data is made. Then, when changes to the volume are made, the blocks of data that are modified are read and indexed to a ”differences area” which will preserve a copy of the blocks prior to any data being overwritten. The blocks indexed in the differences area and the original volumes allows a logical representation of the shadow copy volume at the point in time it was snapshotted.

3 *shadowcopy.py*

shadowcopy.py is a python-based command-line tool developed to process shadow volumes contained in Virtual Hard Disk (VHD) images. When invoked on an image, it will enumerate all shadow volumes on that disk, walk each volume and the files contained, and do at least one of the two following things:

1. Hash the file for reporting purposes.
2. Extract every file walked to the full original path.

When extracting from multiple shadow volumes, it will deduplicate files based on hash. If a file is encountered with the same file name as a previously walked file but different hash, then that file is renamed with a three digit *XXX* delimiter to indicate the difference.

When producing the report, a tab-delimited .txt file is created containing all file names, the originating shadow volume the file came from, the file size, and md5 hash values. This file can be imported into a spreadsheet program for further viewing.

3.1 Requirements

shadowcopy.py has the following base requirements for it to function properly:

- Microsoft Windows Vista, 7 or later
- Python 3.2: If later versions come out, these may or may not work depending on syntax and library revision. Earlier versions of python will not work.
- Administrator Access: *shadowcopy.py* requires the end-user to have access to the “Administrator” account on the system being used to inspect the shadow volumes. If the program is run, it might, and generally will, ask for the administrator password to run. This is a requirement for the build-in command-line tools Microsoft provides.
- vhdtool.exe: A tool that converts raw/dd images into VHD images, if the image desired to be processed is not in a VHD format.

3.2 Features

The development of *shadowcopy.py* was dictated by fundamental features that were seen as the the most important to accomplish as automated of an analysis as possible within the given time frame.

For our purposes, the most important was the ability to automatically enumerate all shadow volumes and proceed to walk each volume hashing each encountered file. The constraint was to ensure only built-in python libraries were used in order to ensure *shadowcopy.py* would be usable across any platform that had the same version of python available.

A concurrent priority was outputting that information into a format that would be human readable. Generally speaking, attributes that were found desirable were to know the names of each file, the full path location of each file, the originating shadow volume, the size of each file, the hash value calculated for each file, and timestamps of the latest modification time, last access time, and creation time of the file (i.e., the file's MAC times).

Other features include choosing between the local (host) machine or everything else, and extracting all files or just providing report. An initial assumption was made that the tool would be useful to run on a machine under inspection and analysis, if needed. The next logical assumption made was disk images are more commonly processed and analyzed, so the default behavior of *shadowcopy.py* is for it to choose all shadow volumes that don't belong to the local machine.

4 Existing Tools

We conducted a brief survey of tools currently available to extract files from Microsoft shadow volumes. Among our findings:

1. Few commercial tools exist that perform a manual analysis.
2. No open source tools publicly exist that perform the capabilities we desired.
3. We could not find tools that are easily extensible for future feature additions.
4. Current tools right now are not architected to automate the extraction, deduplication, and hashing process.

4.1 Commercial Tools

Commercial tools allegedly exist to perform shadow volume forensics but we had trouble acquiring the actual software. When we tried to acquire these tools for testing, one was unavailable despite us contacting the vendor [8], and two appear still be under development [3, 9].

Additionally, the information provided by all three vendors make a strong implication that the their software is targeted to the forensics examiner to manually analyze a disk image.

4.2 Open Source / Commercial Tools Hybrid Approaches

There is no single open source tool in the public domain that perform an automated extraction, deduplication, and hashing of all files in the residing shadow volumes on a disk image. During the survey, it was seen that many forensics practitioners resorted to using a combination of different tools to accomplish some of the capabilities we desired.

Some approaches found to analyze shadow volumes was to use *dd* [13] to image an individual shadow volume, and then perform additional processing with other tools. One approach used the commercial tool *Encase* with their Physical Disk Emulator (PDE) module [11] to perform a manual forensics examination [2]. A second approach used *ntfs-3g* [12] to mount the dd-created image for inspection [4].

A method described to create shadow volume timelines and extract unallocated space from these volumes was to utilize command-line tools from *Sleuthkit* [1, 5]. First, the shadow volumes would have to be enumerated, then the *fls* command would be used to list all the files in the volume. Then, the *mactime* command would be used to create the timeline. To extract unallocated space, the *blkls* command would be used on a shadow volume.

While these approaches have merit, the problem is that they are all manually drives which reduces the productivity of a forensics examiner. Further, some of these methods require more than one operating system platform to be used in order to accomplish the objective.

5 Test and Evaluation

We tested our tool on an image we constructed to verify that it was functioning correctly and then expanded usage into other data. This is a fair test and evaluation because we could use other methods to verify our "ground truth" when performing a post-processing evaluation on our tool's results.

The following section presents the testing portion of *shadowcopy.py* and a discussion on the results and limitations encountered.

5.1 Testing Environments

There were two environments used for testing *shadowcopy.py*. Initially, development and testing was done on a machine running Apple Mac OSX with VMware running a Windows 7 Virtual Machine (VM). An external hard drive containing a test VHD converted from a known VM was created for the "ground truth" image.

When testing, the tab-delimited digest of files was compared against a manual find of files, including hidden and system files, on the VHD performed using the standard Microsoft Windows find and locate files command. Several samples were taken, either by size constraints, or by inspection of directory listings.

The second testing environment was a Windows 7 Desktop running a Windows 7 VM. The "ground truth" image was copied over to ensure results would align with the original testing results from the first environment.

5.2 Testing Results

As expected, both environments performed identically in walking the "ground truth" image, enumerating the shadow volumes, and digesting all files that encountered. Because the python libraries are identical, there is no reason to believe that results would differ.

5.3 Known Limitations

There are some current limitations to the tool that preclude the claim of it being completely automated. First, the process of automatically mounting the VHDs is possible by command-line using *diskpart* but the exact process at the time of writing this is unknown. This is an immediate addition that will be implemented into the next version release of *shadowcopy.py*.

During testing, the authors noted that VHDs would not mount over a network share. While researching this problem, it was discovered that VHDs cannot be attached when stored on a network file system (NFS) or File Transfer Protocol (FTP) server [6]. [6] mentioned that VHDs located on a Server Message Block (SMB) share can be attached. VMware shared folders operate over *Samba/Common Internet File System (CIFS)* so attaching VHDs should have worked but did not. The immediate known known solution was to simply attach an external hard drive to the host machine so that the external hard drive would be made available locally to the VM.

Third, despite the script running as "Administrator" in Windows, there were still files that were being walked that could not be read and hashed. These files are presumably critical system files that may need to be walked as "System".

6 Conclusions

We have shown that we can implement a completely python-based tool to enumerate shadow volumes on a disk image and digest all the files contained in those shadow volumes. We have also shown that the tool has the potential to be completely automated in an environment where disk images can be automatically fed into a framework for analysis during ingestion.

6.1 Future Work

The initial effort on this tool suggests many avenues for extensibility. Some desired additions to the implementation include the following:

1. Implement an automatic mounting process with Microsoft's built-in *diskpart* tool.
2. Choosing specific shadow volumes rather than enumerating and walking all shadow volumes found.
3. Deduping against known hashes (e.g., NSRL repository). If this was performed against every encountered image, this could result in a substantial increase in analysis overhead.
4. Export files by file type. This would involve implementing carving techniques if reading file headers/footers or integrating existing carving technology into the process chain.
5. Similarity digest hashing (SDHash) of walked files as proposed by Roussev [10].
6. Implement a multi-threaded version of this to increase efficiency.
7. Integrate a faster hashing system (e.g., md5deep) over the built-in python implementation.

6.2 Acknowledgements

We wish to acknowledge Brian Madden for his initial work that led to shadowcopy's development.

A Program Listing

Both of the programs listed here are embedded in the PDF of this technical report.

A.1 shadowcopy.py

```
""" shadowcopy command line tool """

# 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'

import sys,os,glob,platform,ctypes
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())


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
    global options
    for (dirpath,dirnames,filenames) in os.walk(v.volumePath()):
        print(”{}”.format(dirpath))
        for filename in filenames:
            shadow_fn = os.path.join(dirpath,filename)
            try:
                st = os.stat(shadow_fn)
                if options.minsize <= st.st_size <=options.maxsize:
                    with open(shadow_fn,”rb”) as f:
                        map = mmap.mmap(f.fileno(),length=0,access=mmap.ACCESS_READ)
                        md5 = hashlib.md5(map)
                        if md5.digest() not in seen:
                            # This file hasn't been seen before.
                            # Write the file's info to the report and copy it to the dest
                            original_fn = shadow_fn.replace(v.volumePath(),””);
                            dest_fn = shadow_fn.replace(v.volumePath(),destdir)
                            dest_fn = make_filename_distinct(dest_fn)
                            print(original_fn,
                                    md5.hexdigest(),
                                    os.path.getsize(shadow_fn),
                                    v.originatingMachine(),
                                    v.volumeName(),
                                    dest_fn,sep='\t',file=report)

                            # If we are extracting, make the directories and copy the dat
                            if not options.noextract:
                                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())
            except (WindowsError) as ex:
                print("Windows cannot read: {}; \n{} continuing...".format(shadow_fn,str(
                print("\t".join([shadow_fn, str(ex)]),file=report)
                continue
            except (IOError) as ex:
                print("Windows cannot open: {}; \n{} continuing...".format(shadow_fn,str(
                print("\t".join([shadow_fn, str(ex)]),file=report)
                continue



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

    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.

"""
    parser.add_option("--list",help="Show the shadow volumes that are avaialble.",
                      action="store_true")
    parser.add_option("--local",help="Analyze only the local machine",
                      action="store_true")
    parser.add_option("--image",help="Analyze a selected image (converts if necessary to 
                      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.txt")
    parser.add_option("--zap",help="Overwrite report file if it exists",
                      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():
    print("This script must run as the Windows Administrator")
    exit(1)

# If the c:\vhd directory does not exist, create it.
#if not os.path.exists(VHD_DIR):
#     os.path.mkdir(VHD_DIR)
# If there are any mounted items in the directory, unmount them

# Get all volumes (local and non-local) for listing
vols = ShadowVolume2.availableVolumes()
if(options.list):
    include_legend = {True:"+",False:" "}
    fmt ="{:1} {:10} {:25} {}"
    print(fmt.format("","Source", "Creation Time", "Volume Name"))
    print(fmt.format("","———", "——————", "—————"))
    for v in vols:
        print(fmt.format(include_legend[include_volume(v,options.local)],
                                            v.originatingMachine(),v.ctime(),v.volumeN
    print("")
    print("+ means volume will be included in analysis")
    exit(0)

seen = set()               # for seen MD5 codes
destdir = ""
if not options.noextract:
    if len(args)!=1 :
        # Demand that we did not get an extract dir
        print("No extraction directory provided.\n")
        parser.print_help()
        exit(1)
    destdir = args[0]

if os.path.exists(options.reportfn) and not options.zap:
    print("{} exists. Delete it or specify a new report filename with --report option
    exit(1)


# Unmount all the mounted images

# Walk and digest only the non-local shadow volumes
```

```
    # Process all of the files
    report = open(options.reportfn, 'w',encoding='utf-8')
    print("\t".join(['Path','MD5','Size','Machine','Volume','Destination Filename']),file
    for v in vols:
        if not include_volume(v,options.local):
            continue
        print("Processing {} from {}".format(v.volumeName(),v.originatingMachine()))
        process(seen,destdir,v,report)
```

A.2 ShadowVolume2.py

```
# Code taken from Brian Madden
import re
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 frmo 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 ")
    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:"):
            current = {}        # start of the new one
            id = line.split(":")[1][1:]
            current['id'] = id
            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:]
            if name.endswith("creation time"): name="creation time"
            current[name] = value
    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_exec(cmd):
    """Run diskpart.exe with cmd and return the results as a string array"""
    p = Popen(['diskpart'],stdin=PIPE,stdout=PIPE);
    p.stdin.write(cmd+"\n")
    p.stdin.close()
    return p.communicate()[0].decode('utf-8').splitlines()
```

B User Documentation

shadowcopy.py is a tool that extracts and de-duplicates files from multiple shadow volumes contained within a Windows Vista or Windows 7 NTFS file system. It produces a directory containing all of the files and .txt file containing all file names, the shadow volume from which they came, their lengths and hash values.

Microsoft Shadow volumes are part of the Volume Snapshot Service (VSS) that are used by the New Technology File System (NTFS) to implement system restore points. System restore points are created automatically when new software is installed or manually when requested by the user.

shadowcopy.py is implemented as a Python 3 script that calls command-line tools are provided by Microsoft as part of Windows. These tools allow a user to access the shadow volumes located on the local drive or on disk images that are in the Microsoft Virtual Hard Drive (VHD) format.

shadowcopy.py can also convert a raw disk image into the VHD format using the *vhdtool.exe* utility which can be downloaded from the Microsoft website.

C Requirements

The following are requirements for *shadowcopy* to function properly.

- Microsoft Windows Vista, 7, or later
- Python 3.2 / 3.2.1 (If later versions come out, these may or may not work depending on syntax and library revision)
- vhdtool.exe: A tool that converts raw/dd images into VHD images.

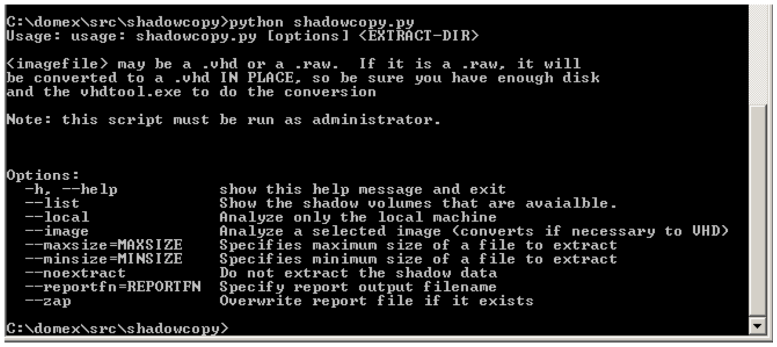
- Administrator Access: The tool requires the end-user have access to the ”Administrator” account on the system being used to inspect the shadow volumes. If the program is run, it might(and generally will) ask for the administrator password on the system to run. This is a requirement for the built-in command-line tools Microsoft provides.

D How To Use

The following section describes how to use shadowcopy.

D.1 The Command Line Menu

shadowcopy is a command line tool and does not have a GUI (later versions may incorporate a GUI). The following is a screenshot of the program output when *shadowcopy* is invoked with no options.



```
C:\domex\src\shadowcopy>python shadowcopy.py
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
  --list                Show the shadow volumes that are avaialble.
  --local               Analyze only the local machine
  --image               Analyze a selected image (converts if necessary to VHD)
  --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

C:\domex\src\shadowcopy>
```

Figure 1: The command-line output of shadowcopy in a Windows 7 shell

As mentioned in the requirements, the images *shadowcopy* requires are VHDs but if the user can only provide a raw/dd image, then *shadowcopy* will automatically convert the image to a VHD prior to any other operation.

D.2 Modes of Operation

shadowcopy has two main modes of operation: walking shadow volumes on the local machine or walking shadow volumes on a VHD that is mounted. The following syntax are examples that will invoke an instance of using either mode:

Local Machine *python shadowcopy.py –local –noextract –reportfn=”c:\some\report\path\ name\report.txt”*

Mounted VHD *python shadowcopy.py –noextract –reportfn=”c:\some\report\path\ name\report.txt”*

The following is a list of the options available for use:

- –list: Shows the shadow volumes that are available both locally and non-locally (mounted VHD)
- –local: Option to only analyze the local machine shadow volumes (generally not needed)
- –image: Option to analyze a specific image by specifying the VHD filename. This option will also convert the image if it is still a .raw/.dd file.
- –maxsize=MAXSIZE: Specifies a maximum file size in **bytes**
- –minsize=MINSIZE: Specifies a minimum file size in **bytes**
- –noextract: Option flag to **not** extract all files being walked
- –reportfn: Specifies a report filename to output results to. This includes the file names, the shadow volume from which they came, their lengths and hash digests.
- –zap: Option to overwrite the report file if it already exists.

D.3 Mounting a VHD for walking

shadowcopy currently does not use automatic mounting of VHD files. The current workaround is to manually mount a VHD file which is described as folows.

1. Right-click on Computer and select ”Manage”.

2. In the computer management window, select ”Disk Management” from the ”Storage” tree on the left-most menu. Once there, go to the ”Actions” menu on the right side and select ”More Actions”. This will bring up a dialog for VHD management. Select ”Attach VHD”.

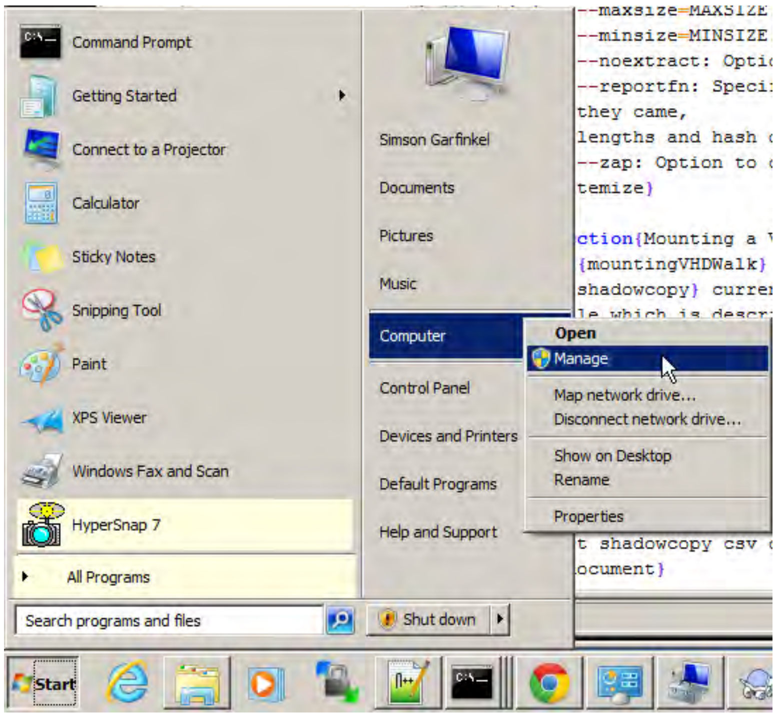
3. In the ”Attach Virtual Hard Disk” dialog, select the .vhd desired.

4. Once selected, check the ”Read-only” box.

From there, you can refer back to Section D.2 to walk the mounted VHD shadow volumes for processing and analysis.

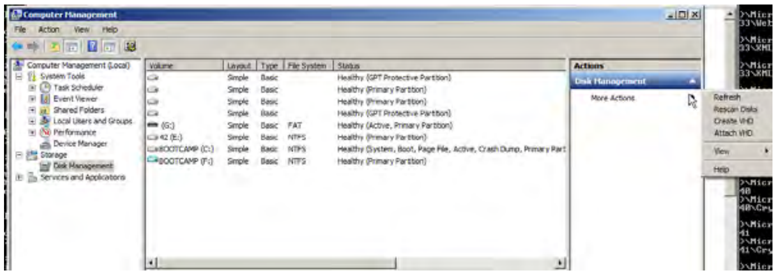
D.4 shadowcopy Output

shadowcopy will output a digest in a tab-delimited text file digest. This allows the end-user to see all files that have been contained in the shadow volumes, md5 hashes, and the full path the file was located in on the filesystem.



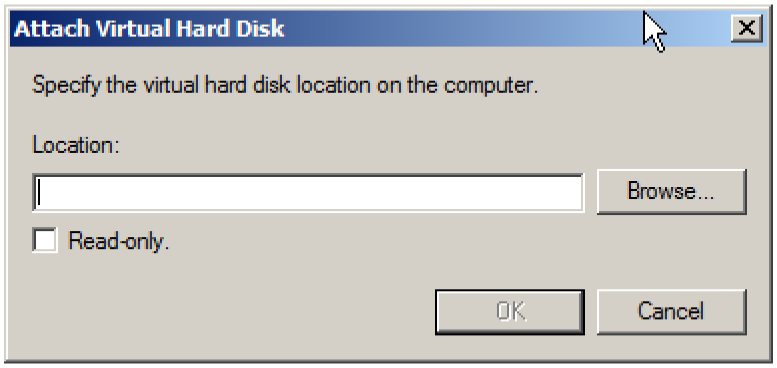
```
--maxsize=MAXSIZE
--minsize=MINSIZE
--noextract: Optio
--reportfn: Speci
they came,
lengths and hash
--zap: Option to
temize}

ction{Mounting a
{mountingVHDWalk}
shadowcopy} curre
le which is descr

t shadowcopy csv
ocument}
```

Figure 2: The context menu when right-clicking on Computer



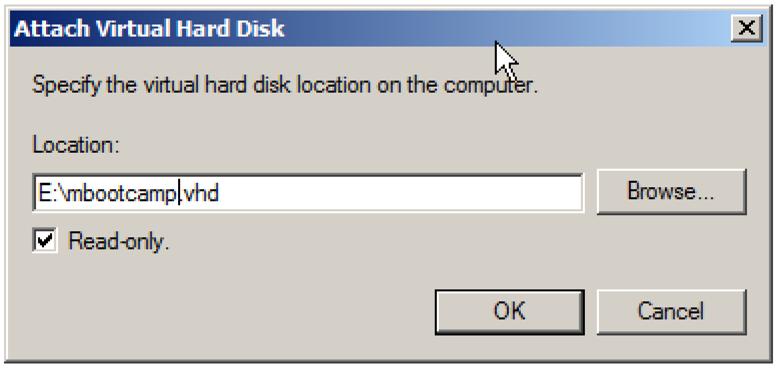
Volume	Layout	Type	File System	Status
	Simple	Basic		Healthy (GPT Protective Partition)
	Simple	Basic		Healthy (Primary Partition)
	Simple	Basic		Healthy (Primary Partition)
	Simple	Basic		Healthy (GPT Protective Partition)
(G:)	Simple	Basic	FAT	Healthy (Active, Primary Partition)
42 (E:)	Simple	Basic	NTFS	Healthy (Primary Partition)
BOOTCAMP (C:)	Simple	Basic	NTFS	Healthy (System, Boot, Page File, Active, Crash Dump, Primary Part
BOOTCAMP (F:)	Simple	Basic	NTFS	Healthy (Primary Partition)

Figure 3: The context menu when clicking on "More Actions" under the "Disk Management" menu in Computer Management



Specify the virtual hard disk location on the computer.

Figure 4: The Attach Virtual Hard Disk dialog


Attach Virtual Hard Disk

Specify the virtual hard disk location on the computer.

Location: E:\mbootcamp.vhd

Figure 5: The Attach Virtual Hard Disk Dialog with "Read-only" checked

References

[1] Brian Carrier. The Sleuth Kit and Autopsy: Forensics tools for Linux and other Unixes, 2005. URL `http://www.sleuthkit.org/`.

[2] Richard Drinkwater. Volume shadow copy forensics.. cannot see the wood for the trees?, 2010. URL `http://forensicsfromthesausagefactory.blogspot.com/2010/02/volume-shadow-copy-forensics-cannot-see.html`.

[3] Sanderson Forensics. Computer forensics software - reconnoitre - vsc, 2011. URL `http://www.sandersonforensics.com/content.asp?page=588`.

[4] Rob T Lee. Vista and windows 7 shadow volume forensics, 2008. URL `http://computer-forensics.sans.org/blog/2008/10/10/shadow-forensics`.

[5] Rob T Lee. Shadow timeslines and other volume shadow copy digital forensics techniques with the sleuthkit on windows, 2010. URL `http://computer-forensics.sans.org/blog/2010/03/16/shadow-timelines-and-other-shadowvolumecopy-digital-forensics-techniques-with-the-sleuthkit-on-w`

[6] Microsoft. Frequently asked questions: Virtual hard disks in windows 7 and windows server 2008 r2, 2010. URL `http://technet.microsoft.com/en-us/library/dd440865(WS.10).aspx`.

[7] Microsoft. Backup: Volume shadow copy service, 2011. URL `http://msdn.microsoft.com/en-us/library/bb968832(v=VS.85).aspx`.

[8] PATCtech. Ekl software shadow scanner, shadow volume analyzer, 2011. URL `http://www.patctech.com/forensics/utilities/eklshadow.shtml`.

[9] Technology Pathways. Volume shadow copy with prodiscover, 2011. URL `http://toorcon.techpathways.com/uploads/VolumeShadowCopyWithProDiscover-0511.pdf`.

[10] Vassil Roussev. *Data Fingerprinting with Similarity Digests*, pages 207–225. Springer, 2010.

[11] Guidance Software. Guidance software encase, 2011. URL `http://www.guidancesoftware.com/forensic.htm`.

[12] Tuxera. Ntfs-3g + ntfsprogs, 2011. URL `http://www.tuxera.com/community/ntfs-3g-download/`.

[13] Wikipedia. dd (unix), 2011. URL `http://en.wikipedia.org/wiki/Dd_(Unix)`.

Initial Distribution List

1. Research and Sponsored Programs Office, Code 41
   Naval Postgraduate School, Monterey, CA 93943
2. Defense Technical Information Center
   Ft. Belvoir, Virginia
3. Dudly Knox Library
   Naval Postgraduate School
   Monterey, California
4. GDS Program Office, Defense Information Systems Agency
   Fort Huachuca, AZ
   `gds@disa.mil`