

Windowless Shadow Snapshots

Analyzing Volume Shadow Snapshots (VSS) without using Windows

by Joachim Metz

Abstract

Had enough of going through a lot of effort just to access the Volume Shadow Snapshots (VSS) directly from an image? The aim of the libvshadow project is to provide access to VSS format directly from a RAW image. This paper will cover a high level overview of the VSS format and discuss various consequences for forensic analysis. It will also discuss how the vshadowtools can be applied in forensic analysis.

Note that the findings in this paper are as-is, for a more complete and up to date understanding of the VSS format see the vshadow project [<http://code.google.com/p/libvshadow/>].

Document information

Title: Windowless Shadow Snapshots
Subtitle: Analyzing Volume Shadow Snapshots (VSS) without using Windows
Author(s): Joachim Metz <joachim.metz@gmail.com>
Date: October 2012
Keywords: VSS, Volume Shadow Snapshot, Volume Shadow Copy Service

Table of contents

- [Abstract](#)
- [Document information](#)
- [Table of contents](#)
- [Introduction](#)
 - [Snapshots on Windows](#)
 - [vshadowinfo](#)
 - [vshadowmount](#)
 - [Format analysis and troubleshooting](#)
- [Snapshots on-disk](#)
 - [The catalog](#)
 - [The store](#)
 - [The store information](#)
 - [The store current and previous bitmap](#)
 - [The store block list and block range list](#)
 - [Stacked snapshots](#)
 - [Blocks, more blocks, oh no even more blocks](#)
 - [The overlay](#)
 - [The forwarder](#)
 - [Reverse block mappings](#)
 - [Other flags](#)
 - [Unused snapshot-volume space](#)
 - [Testing, a great opportunity to find strange behavior](#)
- [Conclusion](#)
 - [Tracking changes across snapshots](#)
 - [Inter-snapshot analysis](#)
 - [Various resources on Volume Shadow Snapshots](#)

Introduction

Volume Shadow Snapshots (VSS) is a Windows specific technology to make backups (or copies) of data on a volume. It was introduced in Windows Vista and its use seems to be limited to volumes that contain the NTFS file system. VSS is part of a larger framework referred to as Volume Shadow Copy Service. Versions of Windows pre-dating Vista, namely XP and 2003, already used the Volume Shadow Copy Service, to make backups and maintain multiple versions of files on remote shares, or to allow access to files for backup purposes that are locked by Windows, e.g. the Exchange database. For sake of clarity the abbreviation VSS can also refer to Volume Shadow Copy Service and Visual Source Safe, in this document however VSS refers to Volume Shadow Snapshots.

VSS is different from the pre-Vista equivalent in regards that the changes are now maintained locally on-disk. On Windows Vista and Windows 7 system volume copies are made automatically and also integrate a replacement for the Windows XP restore points. According to [TECHNET-VISTA] Windows Vista typically makes a snapshot once per day. Windows 7 seems to have reduced this to once a week [SZYNALSKI09].

VSS is a useful feature for forensic analysis and data recovery, because a disk will now contain multiple versions of files and information within them. Since its introduction several papers and articles about using VSS in forensic analysis have been written (see: References). Although there is a bias in most of these papers and articles, nearly all of them focus on utilizing the functionality in Windows itself and not on leveraging the on-disk format.

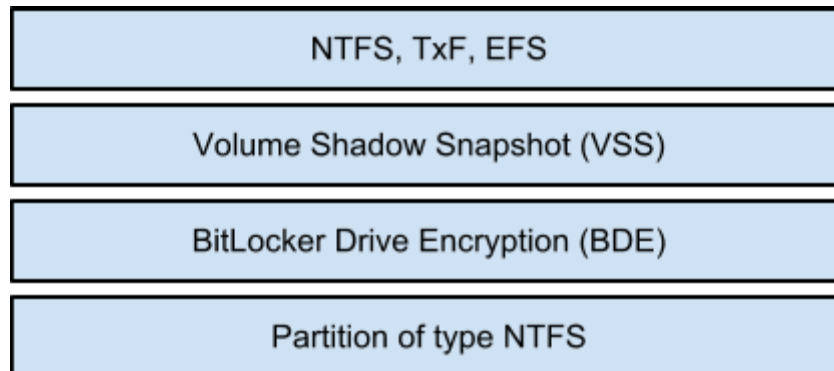
Documentation of the technical details of VSS do not seem to exist in the public domain. In 2011 the libvshadow project [LIBVSHADOW] was started with the goal to analyze and document the VSS on-disk format. This would ultimately allow the creation of tooling to analyze the VSS snapshots without having to rely on the functionality built into Windows.

The rest of this chapter will provide for a brief overview of VSS snapshots on Windows. The second chapter will describe how the vshadow library and tools can be utilized in forensic analysis and data recovery. The third chapter will go into detail how the on-disk format is structured and is the largest part of this paper. The last chapter will discuss various smaller related subjects like relevant findings, implications and future research.

Note that the findings in this paper are as-is, for a more complete and up to date understanding for the VSS format see [LIBVSHADOW].

Snapshots on Windows

Windows maintains the snapshots transparently from the file system. On a “regular” Master Boot Record (MBR)- / GUID Partition Table (GPT)-partitioned disk VSS would logically be in between the filesystem and lower-level subsystems like BitLocker encryption (BDE).



Most of the Volume Shadow Snapshot (VSS) subsystem is inside the kernel-space driver, named volsnap.sys. The Volume Shadow Copy Service and thus VSS has an API in user-space, which largely communicates with the driver by IO control (ioctl) commands. There is no public documentation available from Microsoft about the inner workings of the driver, the IO control and the actual on-disk format.

From user-space the snapshots can be queried via “Previous Versions” in Windows Explorer or via one of the tools that Windows provides to manage the snapshots, like “vssadmin” or “diskshadow”. E.g. to list the snapshots (shadows in vssadmin terminology) with vssadmin:

```
vssadmin list shadows

vssadmin 1.1 - Volume Shadow Copy Service administrative command-line
tool
(C) Copyright 2001-2005 Microsoft Corp.

Contents of shadow copy set ID: {39a136a8-0998-4e24-be64-1a4205ee199b}
  Contained 1 shadow copies at creation time: 23-2-2008 23:19:49
    Shadow Copy ID: {e72ce60b-0e46-4704-aa50-49162700e55c}
      Original Volume: (C:)\?\Volume{ddb2011b-9099-46f1-9d5e-
9b4196327030}\
    Shadow Copy Volume: \?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1
      Originating Machine: hostname
      Service Machine: hostname
      Provider: 'Microsoft Software Shadow Copy provider 1.0'
      Type: ClientAccessibleWriters
      Attributes: Persistent, Client-accessible, No auto release,
Differential, Auto recovered
```

As the output hints Windows provides a device file for the snapshot namely:
“\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1 (or its
alias “\\.\HarddiskVolumeShadowCopy1”).

These device files can be imaged, or analyzed with file system forensic tools like the SleuthKit. Note that currently only a few forensic tools seem to offer support to access these device files. Some of these forensic tools are: Shadow Explorer, ProDiscover, X-Ways Forensics, VSC Toolset and recently EnCase with VSS Examiner EnScript¹.

One noteworthy fact about these device files is that they are non-buffered, meaning that every read must be sector-aligned. This behavior still prevails even when the Windows API CreateFile function is called without the FILE_FLAG_NO_BUFFERING flag.

Beside the device files Windows also maintains several files related to VSS, e.g.

C:\System Volume Information\{3808876b-c176-4e48-b7ae-04046e6cc752}
C:\System Volume Information\{%GUID%\{3808876b-c176-4e48-b7ae-04046e6cc752}

Where %GUID% is a random GUID related to the snapshot and C: is an example of the “drive letter” of the corresponding volume.

Because the snapshots are volume-based these files were most likely created to prevent non-VSS aware versions of NTFS from overwriting the VSS data on the volume.

It is common practice when performing forensic analysis of VSS to have Windows deal with the structures, by either making a full volume image of an individual snapshot-volume, or a live analysis of the snapshot-volume by forensic file system tools. There are downsides to this approach since it not publicly known how the VSS structures should be interpreted and how this is done by different versions of Windows.

One example of this is that according to [MSDN-VSS] VSS can hide volumes:

The shadow copy is locally exposed. If this bit flag (VSS_VOLSnap_ATTR_EXPOSED_LOCALLY) and the VSS_VOLSnap_ATTR_EXPOSED_REMOTELY bit flag are not set, the shadow copy is hidden.

However tests on Windows 7 have shown that vssadmin still lists the shadows even though both bits were not set. The third chapter will go further into detail of the VSS behavior.

Additionally in the case of running an acquired system in a VM, any software on the system can influence access and reading the snapshots, which should not be surprising to the reader.

Vshadow library and tools

Currently the libvshadow source package comes with a library and tools. The library offers a stand-alone implementation of VSS functionality and can be integrated into other tools. At the time of writing the library is considered alpha status and can be used without any guarantees. The alpha status means that the library is undergoing thorough testing and core

¹ See: http://www.forensicswiki.org/wiki/Windows_Shadow_Volumes for the most up to date list of tools.

functionality is susceptible to change.

At the moment the libvshadow package comes with two tools, namely vshadowinfo and vshadowmount, which will be discussed briefly in the following paragraphs.

vshadowinfo

The vshadowinfo tool can be considered the vshadow counterpart of vssadmin. Where vshadowinfo only will show you information about the Volume Shadow Snapshots and does not have functionality to manage them.

Vshadowinfo expects a raw volume data as input. So in case of an Expert Witness Compression Format (E01) image file the raw image data needs to be made accessible first or in case of a BitLocker encrypted volume it needs to be decrypted first.

Vshadowinfo allows you to specify an offset (in bytes) where the volume is located. This can be useful when running the command against a full disk image. E.g. the volume at sector 2048 (or byte offset 1048576) of a 30 GB disk image:

```
vshadowinfo -o 1048576 image.raw
```

```
vshadowinfo 20120823
```

```
Volume Shadow Snapshot information:
```

```
Number of stores: 2
```

```
Store: 1
```

```
Identifier           : 93db9c47-bb19-4004-836d-c3c835550b9a
Shadow copy set ID    : 8bc68d0b-9df4-49e0-be4b-725dceaaefc8
Creation time         : May 19, 2012 14:20:35.765721000 UTC
Shadow copy ID        : 04057e11-d2d5-4d9c-8914-ae8a832e467b
Volume size           : 30002905088 bytes
Attribute flags       : 0x00420009
```

```
Store: 2
```

```
Identifier           : 93db9c47-bb19-4004-836d-c3c835550b9a
Shadow copy set ID    : 90e19848-7436-4309-8899-4bbecf05a566
Creation time         : May 19, 2012 16:13:58.750469800 UTC
Shadow copy ID        : 6db2bc5e-9b95-4ed1-a493-091ee7d2e5e6
Volume size           : 30002905088 bytes
Attribute flags       : 0x00420009
```

vshadowmount

The vshadowmount tool can be considered the vshadow counterpart of the device file handles on Windows. Vshadowmount will fuse-mount the individual snapshot-volumes and make them accessible as volume device files.

E.g. to mount the previously described disk image:

```
vshadowmount -o 1048576 image.raw fuse/
```

```
vshadowmount 20120823
```

This can take a while because at the moment all the stores are initialized. In the future this might change into a more on-demand approach.

Vshadowmount does not output anything if successful. The most straightforward way to verify if it has created the volume device files is to list the contents of the mount point, in this case “fuse/”, e.g.:

```
ls fuse/
```

```
vss1 vss2
```

The output indicates that 2 volume device files for the snapshot-volumes are provided, where vss1 is the store indicated as “Store 1” by vshadowinfo, and so forth.

These volume device files can be further analyzed with tools like the SleuthKit, e.g. running fls:

```
fls fuse/vss1
```

```
r/r 4-128-4:      $AttrDef
r/r 8-128-2:      $BadClus
r/r 8-128-1:      $BadClus:$Bad
r/r 6-128-4:      $Bitmap
r/r 7-128-1:      $Boot
d/d 11-144-4:     $Extend
r/r 2-128-1:      $LogFile
r/r 0-128-1:      $MFT
r/r 1-128-1:      $MFTMirr
...
r/r 3-128-3:      $Volume
d/d 36-144-6:     System Volume Information
d/d 256:          $OrphanFiles
```

Format analysis and troubleshooting

The library and tooling can be compiled with debug and verbose output. When invoked with the verbose option (-v) the library and tools will start generating a lot of output on stderr. This feature is intended for format analysis and troubleshooting.

Debug and verbose output can be enabled by running configure with the following options:
`./configure --enable-debug-output --enable-verbose-output`

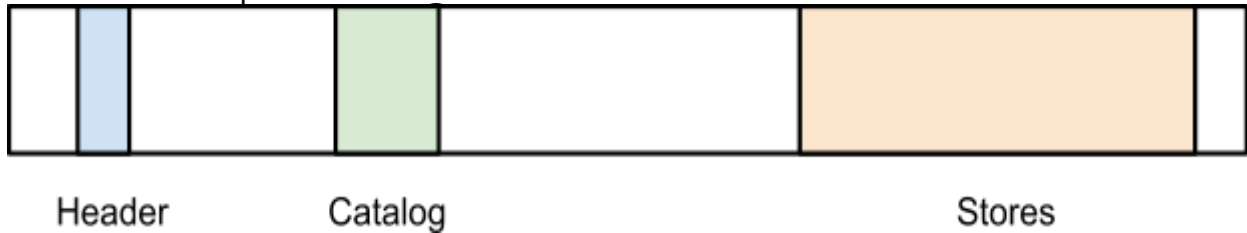
Snapshots on-disk

The on-disk Volume Shadow Snapshot (VSS) structures are largely self-contained and separate from the file system structures. The snapshotting is basically done on volume-level on a 16 KiB block-basis.

The on-disk Volume Shadow Snapshot (VSS) is divided into three main parts:

- The VSS volume header
- The VSS catalog
- The VSS stores

As a schematic representation:



The VSS volume header can be found at offset 7680 (or in hexadecimal 0x1e00) from the start of the volume. The header contains the VSS identifier, which is the GUID: {3808876b-c176-4e48-b7ae-04046e6cc752} and the location (byte offset) of the catalog. The identifier can be used as a signature to search for in case of corruption.

Since the catalog can be located anywhere in the volume it has the corresponding file system entry to protect it from being overwritten by non-VSS aware implementations of NTFS, namely:

`C:\System Volume Information\{3808876b-c176-4e48-b7ae-04046e6cc752}`

Where C: is an example of the “drive letter” of the corresponding volume.

The catalog

The catalog contains information about the stores, like:

- unique identifier (GUID);
- the creation time stored as a 64-bit FILETIME timestamp;
- the location (byte offset) of the store in the volume;
- the corresponding NTFS file reference of its protection-file entry in “\System Volume Information\”.

These store protection-files are named:

`C:\System Volume Information\{%GUID%\}{3808876b-c176-4e48-b7ae-04046e6cc752}`

Where %GUID% is a random GUID related to the snapshot and C: is an example of the “drive letter” of the corresponding volume.

The store

The store contains information about an individual snapshot. Not all of the snapshot data is necessarily stored inside the store. To keep track of where the snapshot blocks are stored the following data structures are used:

- The store information;
- The store (current) bitmap and previous bitmap;
- The store block list and block range list.

The store information

The store information contains some of the data “vssadmin list shadows” normally prints, namely:

- The shadow copy and shadow copy set identifiers;
- The originating and service machine hostnames;
- The provider and the attribute flags.

There are various flags and their meaning can currently only be derived from what information could be found on MSDN [MSDN-VSS]. Note that this source of information seems to be not very clear what part of Volume Shadow Copy Service these affect².

On Windows the number in the device filename is generated by the running system.

The store current and previous bitmap

Every store has a current bitmap. This bitmap indicates which 16 KiB block in the volume is in-use, or “allocated” for lack of a better term. If a bit is set the corresponding block is considered not in-use.

The previous bitmap is not always present, e.g. it is not present in the first store of a newly created volume, but should be present in any other store. The previous bitmap indicates which 16 KiB block in the volume is in-use by the previous volume snapshot. If a bit is set the corresponding block is considered not in-use.

When reading snapshot-volume data the bitmaps seem to be ignored for block ranges that are defined in the block list. This is discussed in more detail further on.

² At the time of writing these flags are ignored by libvshadow.

The store block list and block range list

The store block list and block range list (technically arrays) define block ranges. The store block range list defines which block ranges are used by the store itself. The store block list defines the block ranges that make up the snapshot of the volume. The block list contains block descriptors to define the block ranges.

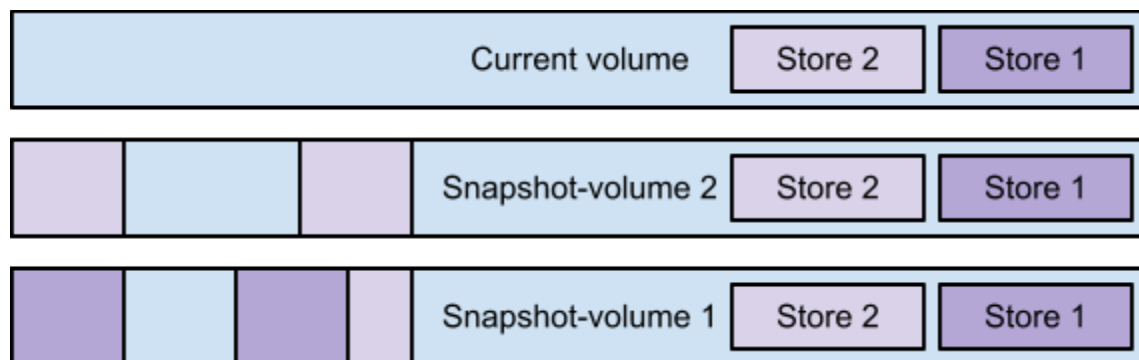
An individual block descriptor consists of:

Byte offset	Byte size	Description
0	8	Original offset The original offset of the data in the volume
8	8	Relative store data offset The offset of the snapshot data relative from the start of the store
16	8	Store data offset The offset of the snapshot data relative from the start of the volume
24	4	Flags
28	4	Optional allocation bitmap

The flags control the behavior of the other values in the block descriptor, which we'll get into more detail after the next paragraph.

Stacked snapshots

At this point it is important to realize that the changes in the snapshots are stacked in reverse-order, meaning that the changes are relative to the current volume. The current volume is the volume as currently available to the system, e.g. "\\.\C:". The oldest snapshot is snapshot-volume 1 and described by store 1, e.g. "\\.\HarddiskVolumeShadowCopy1", the more recent snapshot-volume 2, described by store 2, , e.g. "\\.\HarddiskVolumeShadowCopy2", as illustrated below.

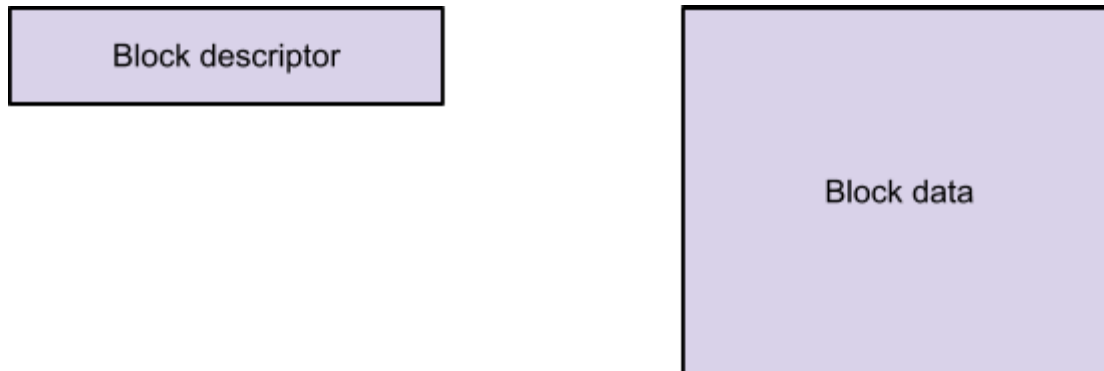


As you can see in the illustration the changes are patched on top of the existing data. So the block descriptors in the store describe how to “patch” the current volume to look like a one of the previous snapshots. To get snapshot-volume 1 both the block descriptors in store 1 and store 2 need to be applied.

Blocks, more blocks, oh no even more blocks

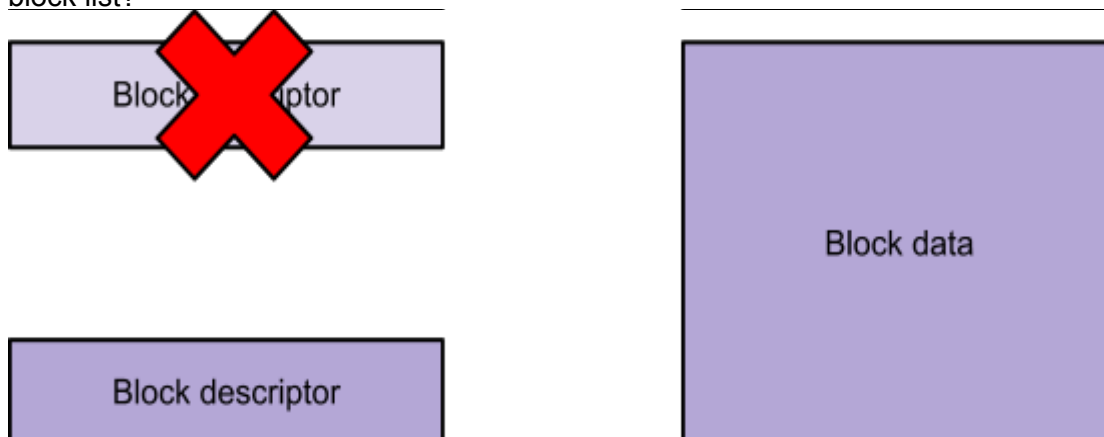
As discussed in previously the block descriptors contain the original offset so that the block containing snapshot data can be mapped back to its original location.

To demonstrate this, let’s consider a current volume with a single store (so one snapshot-volume).



Basically the block data gets “patched” on top of the current volume at the original offset. Commonly the block data is contained within the store. Likely to prevent it from being overwritten.

What if we encounter another block descriptor with the same original offset in the same block list?³

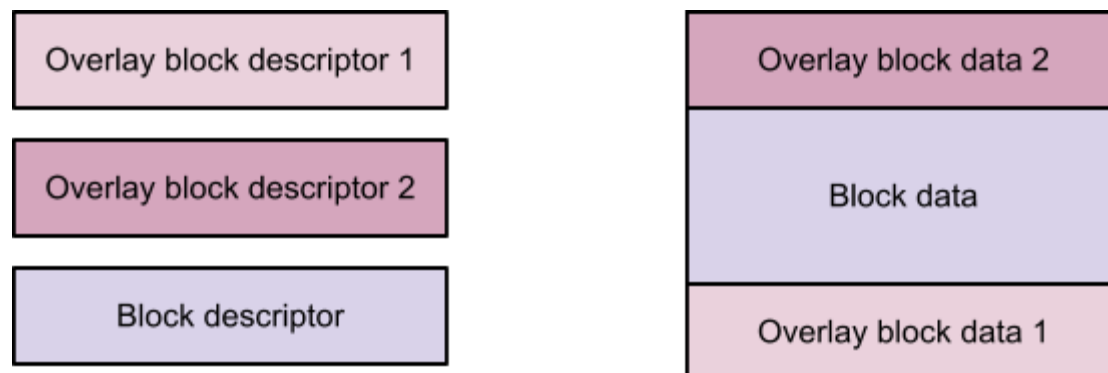


³ This actually happens frequently and seems to part of the design.

In this case previous block descriptor is ignored and the more recent one is applied. Although this data could be useful to determine what was there before the snapshot, for the sake of complexity it is currently ignored. for a potential use case of the ignored block descriptors see the discussion about inter snapshot analysis further in this paper.

The overlay

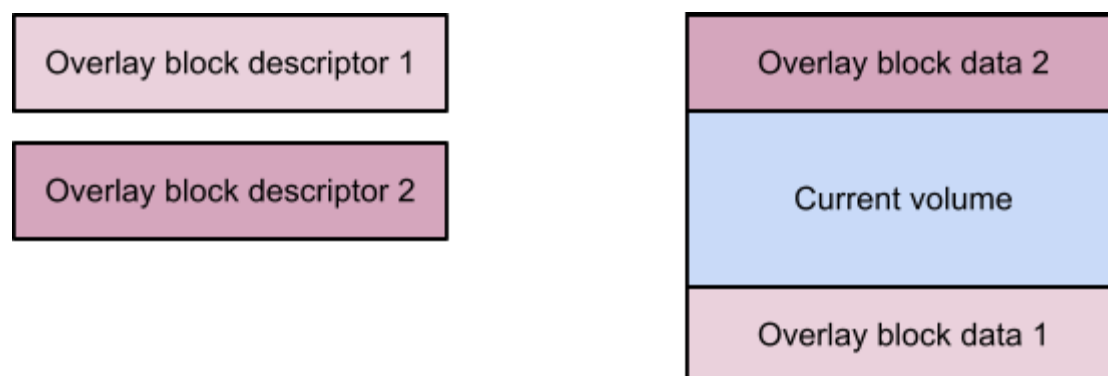
The overlay can maintain changes smaller than 16 KiB. For this the “allocation bitmap” in the block descriptor is used to maintain changes of 512 byte blocks. The block descriptor has the overlay flag (2) set.



As you can see in the illustration:

- The order of the overlay and “normal” block descriptors does not matter. The overlay is applied on top of the “normal” block data.
- There can be more than one overlay block descriptor, although the successive overlay block descriptors should contain a “relative store data offset” of 1 and point to the same “store data offset”.

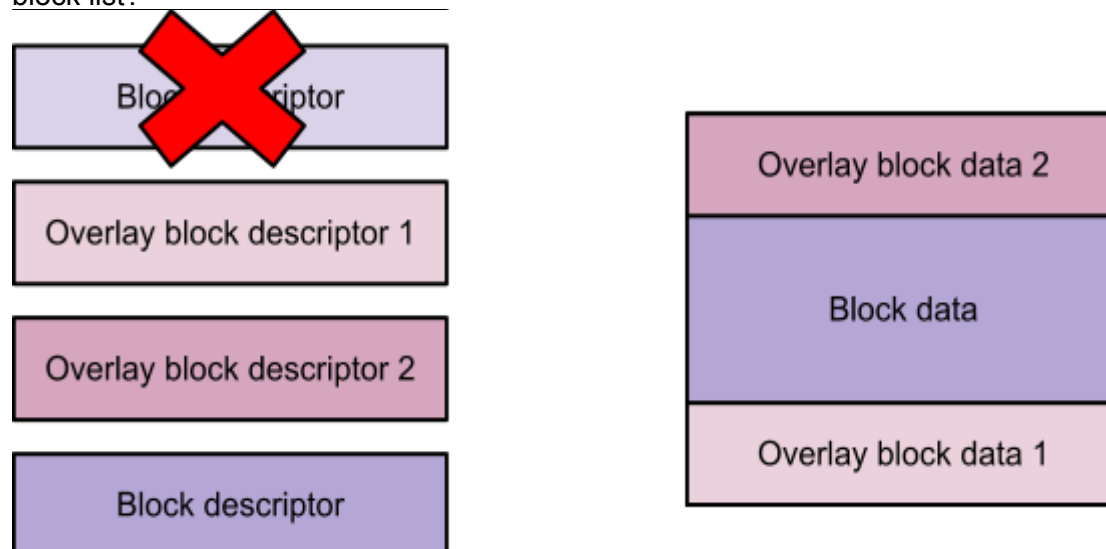
Also a corresponding “normal” block descriptor does not need to be defined, e.g.



In this case the overlay is applied on-top of the current volume (the volume as currently available to the system).

If due to corruption or deliberate manipulation there would be a successive overlay block descriptor defined after overlay block descriptor 2, that points to a different “store data offset”, the Windows implementation considers the new overlay block descriptor part of the previous overlay and therefore the new offset is ignored.

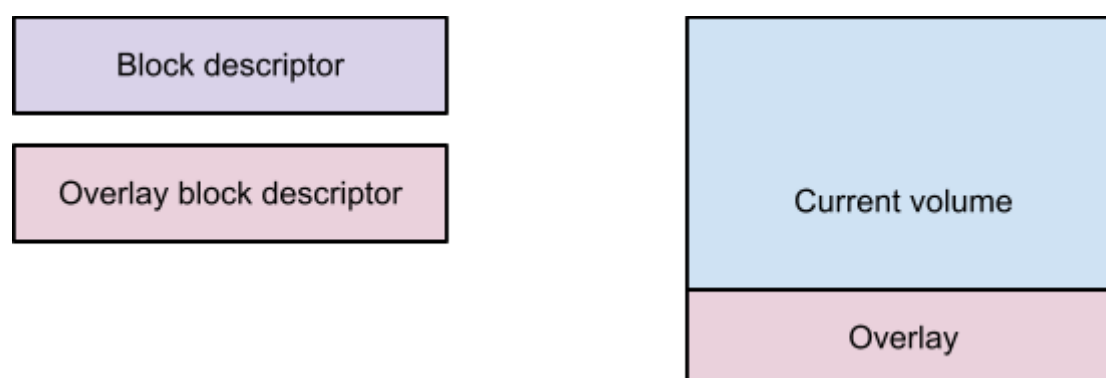
So what if we encounter another block descriptor with the same original offset in the same block list?



As before the previous block descriptor is ignored and the more recent one is applied, but the overlay is still applied to the new block descriptor.

The forwarder

“Forwarded blocks” do not need to be stored inside the store. The block descriptor has the forwarder flag (1) set. The current assumption is that the “forwarded blocks” are used for copy-on-write (COW); meaning that the block is marked as to be copied to the store when the data in the block is overwritten. A forwarder will “forward” the block in the snapshot-volume to a block in the current volume.



Just like for the block descriptor without flags, successive block descriptors overwrite previous block descriptors. Also the overlay is applied on top of the forwarded block.

Reverse block mappings

When forwarders are encountered their reverse block mapping is tracked. E.g. consider the following forwarder block descriptor:

```
original offset: 0x1234000  
relative offset: 0x5678000
```

When this block descriptor is encountered a “forward block mapping” is made from 0x1234000 to 0x5678000, but also a “reverse block mapping” from 0x5678000 to 0x1234000. These reverse block mappings are maintained in a run-time reverse block list.

The reverse block mappings affect the behavior when a successive block descriptor is added, e.g the following:

```
original offset: 0x5678000  
relative offset: 0x9abc000
```

Based on the previously described behavior one would expect 0x5678000 to map to 0x9abc000, but this is not the case. Because a reverse block mapping for 0x5678000 is present the original offset of the new block descriptor is swapped with that of the reverse block mapping, like:

```
original offset: 0x1234000  
relative offset: 0x9abc000
```

The reverse block mapping will be removed for the run-time reverse block list.

If the block descriptor with original offset 0x5678000 would have been a forwarder a new reverse block mapping would have been created, namely one that maps 0x9abc000 to 0x1234000. Just like with forward block descriptors, successive reverse block descriptors overwrite previous reverse block descriptors.

The reverse block mapping has another side-effect that if a reverse block mapping is available the bitmaps are ignored, this will be discussed in more detail in the next paragraph.

Other flags

The forwarder (1) and overlay (2) flags seem to be the most significant regarding block descriptor behavior. Flag 4 is used to indicate if the block descriptor is in use or can be ignored. The other flags often show the same behavior as the "normal" block descriptor (when no flags are set). Currently the actual meaning of the other flags is unknown.

Unused snapshot-volume space

The unused (or unallocated) snapshot-volume space has not been discussed before because its behavior is dependent on the position of the snapshot-volume in the stack. Unused blocks can be zero-filled depending on if they are the most recent snapshot-volume or a snapshot-volume predating the most recent one. From a forensic perspective this data could be more recent than that in the used snapshot-volume space, this behavior can affect the results from file and data recovery techniques applied to the snapshot-volumes.

The behavior for the most recent snapshot-volume:

- If the block offset has a corresponding reverse block descriptor:
 - The block is read from the current volume
- Else if the current bitmap does not have a corresponding entry for the block:
 - The block is read from the current volume
- Else if there is a previous bitmap and it does not have a corresponding entry for the block:
 - The block is read from the current volume
- Else:
 - The block is zero-filled

The behavior of snapshot-volume predating the most recent one:

- The block is read from the current volume

Reading block data

In the previous paragraphs a lot of different factors were described that are involved in how a snapshot-volume is reconstructed, namely:

- The different types of block descriptors.
- The position of the store in the stack.
- The current and the previous bitmaps.

Because these factors are so intertwined the block reading method as described below is not yet considered complete. The current approach seems to hold for most of the test cases. For the most up to date version of the approach see the VSS format specification working document [LIBVSHADOW].

For the size of the data that will fit in the buffer:

- If the block offset has a corresponding block descriptor:
 - The data is defined by block descriptor and has a maximum size accordingly
 - If this is the active store and the block has an overlay:
 - If the overlay applies:
 - use the overlay block descriptor

(continued on next page)

- If the forwarder flag (0x01) is set and there is a next store:
 - read the block from the next store using the relative store offset
 - Else:
 - read the block from the current volume using the store offset
- Else:
 - If there is a next store:
 - read the block from the next store
 - Else if the block offset has a corresponding reverse block descriptor:
 - read the block from the current volume
 - Else if the active store is the most recent (last) store and the block is flagged in the current bitmap and (the store has no previous bitmap or the block is flagged in the previous bitmap):
 - zero the block
 - Else:
 - read the block from the current volume
- Increment the block offset with the size of the block read

Testing, a great opportunity to find strange behavior

Testing is always a great opportunity to find strange, so in the case of VSS. During the analysis of the on-disk format some interesting behavior was seen, namely an inconsistency in how some (forensic) tools read block data from VSS device files on Windows.

Apparently when reading not in-use ranges from the snapshot-volume the buffer passed to the read function is not altered but the read is returned as successful. So the data in these areas is actually undefined and not zeroed as assumed by several tools.

The scenario: a 30 GB hard disk containing 3 snapshot-volumes. The disk is filled with unique markers per 512 byte blocks. Using both Cygwin dd 8.15 and md5sum 8.15, and Forensic Acquisition Utilities (FAU) 1.3.0 dd 5.3.0.

Read a block from the most recent snapshot-volume that is not in-use at offset 1879048192 (or 0x70000000 in hexadecimal, or 16 KiB block number 114688).

```
dd if=\\.\HarddiskVolumeShadowCopy3 of=vss3-0x70000000.bin bs=16384 count=1
skip=114688
```

This returns a zero filled block, which matches the expected behavior.

Read the same block from a less recent snapshot (the block is not in-use by this snapshot-volume either).

```
dd if=\\.\HarddiskVolumeShadowCopy2 of=vss2-0x70000000.bin bs=16384 count=1
skip=114688
```

This returns a block containing the data of the current volume.

Since it's common practice to make an image snapshot-volumes, let's make one of the most recent snapshot-volume. First let's use FAU dd:

```
dd if=\\.\HarddiskVolumeShadowCopy3 of=vss3.raw
```

The resulting image size 30002900992 bytes. The partition is of size 30002905088 bytes and the volume is of 30002900992 bytes, so 4096 bytes or 1 cluster block is hidden. This is the last cluster block and is typical behavior for an NTFS volume on Windows.

Read the same block again, this time from the image:

```
dd if=vss3.raw of=vss3.raw-0x700000000.bin bs=16384 count=1 skip=114688
```

The block data is no longer a zero filled block and points to a total different offset.

Before making conclusions that something is broken, let's do some more tests:

- Calculate the MD5 of the most recent snapshot-volume (device file) before and after imaging. Both MD5 match.
- Calculate the MD5 of the image. The MD5 does not match the MD5 of the device file.
- Try another version of dd (Cygwin) and make another image. Again the resulting image of size 30002900992 bytes. The MD5 does not match with the MD5 of the device file nor with the MD5 of the image created by FAU dd. Although a zero filled block is returned when retrieving the block from the image; coincidence?
- Calculate the MD5 of the most recent snapshot-volume again. Same MD5 as before.

What is going on here? Let's look at the source, which read functions are they using:

1. Cygwin dd uses the read function.
2. Cygwin md5sum uses the fread function.
3. FAU dd uses the read function.

If the cause was solely the read function that does not account for the difference between the MD5 of the image created by FAU dd and Cygwin dd; maybe the block size is also involved?

Let's try to rule out any differences in the read versus fread function and block sizes. By using a variant of md5sum that uses the read function instead of fread and allows to set the block size.

- Calculate the MD5 of the most recent snapshot-volume (before and after imaging) with the read function and a block size 16 KiB. 30002900992 bytes of data is read both MD5 match, the MD5 is different than the one of the fread function-based md5sum.

- Make an image using Cygwin dd and a block size of 16 KiB and calculate the MD5. The MD5 matches that of the device file.
- Calculate the MD5 of the most recent snapshot-volume again with the read function and a block size 32 KiB. 30002900992 bytes of data is read but the MD5 is different.

So the block size seems to be of influence, but how and why?

By doing a 32 KiB block-wise compare of the image and the device file we find that offset 131072 (or 0x00020000) differs. Let's try to analyze some different block sizes.

2 blocks of 16 KiB.

```
dd if=\\.\HarddiskVolumeShadowCopy3 of=2blocks.bin bs=16384 count=2 skip=8
```

1 block of 32 KiB.

```
dd if=\\.\HarddiskVolumeShadowCopy3 of=1block.bin bs=32768 count=1 skip=4
```

256 blocks of 512 bytes.

```
dd if=\\.\HarddiskVolumeShadowCopy3 of=sectors.bin bs=512 count=64 skip=256
```

The 32 KiB block and the 512 bytes blocks match, where the 16 KiB blocks and the 512 bytes blocks do not. The difference occurs after the first 16 KiB.

Let's read each 16 KiB block individually, combine them and compare them again with the 512 byte blocks.

```
dd if=\\.\HarddiskVolumeShadowCopy3 of=2blocks-part1.bin bs=16384 count=1 skip=8
dd if=\\.\HarddiskVolumeShadowCopy3 of=2blocks-part2.bin bs=16384 count=1 skip=9
cat 2blocks-part1.bin 2blocks-part2.bin > 2blocks-concat.bin
```

Strange, now we have a match.

Some additional test indicate that if the block size is enlarged to 64 KiB the 32 KiB block size will also show a difference after the first 32 KiB. So what seems to be happening here is that reading more than the block size results in a "block wrap". But why here and not at earlier offsets? Likely because in this case the current and previous bitmaps flags mark offset 131072 (or 0x00020000) as in-use and offset 147456 (or 0x00024000) as not in-use. Neither offsets are defined by a block descriptor. Although the "block wrap" behavior is strange it seems to be deterministic.

Additional analysis indicates that a similar strange behavior occurs at another point where again the in-use at offset 3221471232 (or 0xc003c000) changes in to not in-use at offset 3221487616 (or 0xc0040000). This time however the 512 bytes blocks show the “block wrap” behavior for the the last sector at 3221487104 (or 0xc003fe00) across the 16 KiB block that follows.

Reviewing the specific offsets in a hexdump program, e.g. xxd, shows no indication of the block wrap.

To determine if this is a bug in either variants of dd, a simple block-wise cat program was created that uses the read function. This program initially showed the same behavior as dd. However when the read buffer was explicitly zeroed before the call to the read function it became clear what the issue was, namely that when reading not in-use ranges from the snapshot-volume the buffer passed to the read function is not altered but the read is returned as successful.

For sanitation reasons libvshadow will assume that these ranges should be zeroed out.

One can argue if this is a bug in the tools or not, nonetheless the “block wrap” behavior has forensic implications. Luckily it seems to be limited to block ranges that are mostly considered unallocated by the file system as well. But if one would carve the snapshot-volume one would find data that is technically non-existent. This also means that it is impossible to verify the data image with the data on the snapshot-volume without knowing and being able to apply the same block size.

Conclusion

In the previous chapters Volume Shadow Snapshots (VSS) and the vshadow library and tools were described in detail. If more detail is required check the VSS format specification working document [LIBVSHADOW].

VSS can be a useful resource in digital forensic analysis, it provides for looking at multiple versions of a file system in time. This can help in understanding changes on the system or to recover data that has been deleted but is still retained in the snapshots.

It also offers new challenges for forensic analysis and tools. First of all, the behavior of the snapshots is (currently) largely unknown and now well understood, only a small number of forensic tools seem to have support for VSS. The tools that do most likely will use the Windows subsystem instead of doing their own parsing and it is unknown how these tools deal with issues like the block wrap. So at the moment analyzing the snapshots largely means doing the same analysis on a different versions of the system. It would be useful if forensic tooling would represent the multiple versions as is done in a versioning system.

Secondly the snapshots can be both a blessing and a nuisance for data recovery. Consider how your carver will deal with multiple versions of the data. If it uses naive carving method, it will generate only more data. On the other hand if the carver (or recovery tool) can incorporate the information maintained in the shadows, it might do a better job of data recovery.

The next paragraphs contain related ideas for future research, some of them can also be found on the "Open Research Topics" page of Forensics Wiki.

Tracking changes across snapshots

At the moment the vshadow project only focuses on providing support for the format. Therefore analyzing multiple versions of a system is still a lot of manual labor. More research is needed to automatically and efficiently map differences and similarities of multiple versions data of the snapshot-volumes on a computer system. The idea is to use the gained knowledge of the VSS format and build more efficient techniques to determine the differences and similarities between snapshots.

Inter-snapshot analysis

As mentioned before, the block descriptors are ignored in favor of later versions. The ignored block descriptors could be used to determine inter-snapshot changes. In other words changes within the snapshot-volume over time. The challenge will be to link these changes on volume-level with changes on file system-level or application-level.

References

[LIBVSHADOW]

Title: Volume Shadow Snapshot (VSS)
Subtitle: Analysis the Windows NT VSS format
Author(s): Joachim Metz
Date: March 2011
URL: <http://code.google.com/p/libvshadow/>

[MSDN-VSS]

Title: Volume Shadow Copy Service Overview
URL: [http://msdn.microsoft.com/en-us/library/windows/desktop/aa384649\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/aa384649(v=vs.85).aspx)

Title: Excluding Files from Shadow Copies
URL: [http://msdn.microsoft.com/en-us/library/windows/desktop/aa819132\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/aa819132(v=vs.85).aspx)

Title: _VSS_VOLUME_SNAPSHOT_ATTRIBUTES enumeration
URL: [http://msdn.microsoft.com/en-us/library/windows/desktop/aa385012\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/aa385012(v=vs.85).aspx)

[FORENSICSWIKI]

Title: Open Research Topics
URL: http://www.forensicswiki.org/wiki/Open_Research_Topics

[TECHNET-VISTA]

Title: Inside the Windows Vista Kernel
URL: <http://technet.microsoft.com/en-us/library/cc748650.aspx>

[SZYNALSKI09]

Title: What you should know about Volume Shadow Copy/System Restore in Windows 7 & Vista (FAQ)
Author(s): Tomasz P. Szynalski
Date: November 23, 2009
URL: <http://blog.szynalski.com/2009/11/23/volume-shadow-copy-system-restore/>

Various resources on Volume Shadow Snapshots

Title: Windows Shadow Volumes
URL: http://www.forensicswiki.org/wiki/Windows_Shadow_Volumes

Title: Vista Volume Shadow Copy issues
Author: DC1743
Date: Monday, 17 August 2009
URL: <http://forensicsfromthesausagefactory.blogspot.com/2009/08/vista-volume-shadow-copy-issues.html>

Title: Accessing Volume Shadow Copies
Author: Harlan Carvey
Date: January 4, 2010
URL: <http://windowsir.blogspot.ch/2011/01/accessing-volume-shadow-copies.html>

Title: Volume Shadow Copy Forensics.. cannot see the wood for the trees?
Author: DC1743
Date: Friday, 19 February 2010

URL: <http://forensicsfromthesausagefactory.blogspot.com/2010/02/volume-shadow-copy-forensics-cannot-see.html>

Title: Into The Shadows
Author(s): Lee Whitfield
Date: April 19, 2010
URL: <http://www.forensic4cast.com/2010/04/into-the-shadows/>

Title: Volume Shadow Copy with ProDiscover®
Date: 2011
URL: <http://toorcon.techpathways.com/uploads/VolumeShadowCopyWithProDiscover-0511.pdf>

Title: Getting Ready for a Shadow Volume Exam
Author(s): Jimmy Weg
Date: June 2012
URL: <http://justaskweg.com/?p=351>

Title: Examining Volume Shadow Copies – The Easy Way!
Author(s): Simon Key
Date: June 2012
URL: <http://encase-forensic-blog.guidancesoftware.com/2012/06/examining-volume-shadow-copies-easy-way.html>

Title: Mounting Shadow Volumes
Author(s): Jimmy Weg
Date: July 2012
URL: <http://justaskweg.com/?p=466>

Title: Examining the Shadow Volumes with X-Ways Forensics
Author(s): Jimmy Weg
Date: July 2012
URL: <http://justaskweg.com/?p=518>

Title: "Weg, I'm afraid that I don't have VMware. How do I Examine Shadow
Volumes?"
Author(s): Jimmy Weg
Date: July 2012
URL: <http://justaskweg.com/?p=710>