

TZWorks® Windows INDX Slack Parser (*wisp*) Users Guide



Abstract

wisp is a standalone, command-line tool used to extract ***INDX*** artifacts from Windows NTFS volumes. ***INDX*** attributes are used to store the contents of a directory. Extracting directory items from the slack portion of the ***INDX*** attribute can identify evidence of a file's past presence after it has been deleted and is no longer part of the system. ***wisp*** can operate on a live volume, an image of a volume or a single directory. All artifacts can be outputted in one of three formats for easy inclusion with other forensics artifacts. ***wisp*** runs on Windows, Linux and macOS.

Copyright © TZWorks LLC

www.tzworks.com

Contact Info: info@tzworks.com

Document applies to v0.59 of ***wisp***

Updated: Apr 25, 2025

Table of Contents

| | | |
|-------|---|----|
| 1 | Introduction | 2 |
| 2 | <i>INDX</i> Attributes (Format/Internals) | 2 |
| 3 | How to Use <i>wisp</i> | 5 |
| 3.1 | Parsing a Live Volume | 6 |
| 3.2 | Parsing an Image File off-line | 7 |
| 3.3 | Handling Directories in Volume Shadows..... | 8 |
| 3.4 | Parsing a NTFS Volume Mounted on Linux or Mac OS-X..... | 9 |
| 3.5 | Parsing a <i>VMWare</i> Volume | 10 |
| 3.6 | <i>wisp</i> Output..... | 11 |
| 3.6.1 | Which Output option yields the most data | 11 |
| 3.6.2 | Two Categories of Slack entries | 11 |
| 3.6.3 | Corruption of the Index Entry | 12 |
| 3.6.4 | Looking at the Raw Data for Slack Entries | 13 |
| 3.7 | Extracting Clusters Associated with a Deleted Entry | 14 |
| 3.8 | Eliminating the Duplicates | 14 |
| 4 | Known Issues..... | 15 |
| 5 | Available Options | 15 |
| 6 | Authentication and the License File..... | 17 |
| 6.1 | <i>Limited</i> versus <i>Demo</i> versus <i>Full</i> in the tool's Output Banner..... | 18 |
| 7 | References | 18 |

TZWorks® INDX Parser (*wisp*) Users Guide

Copyright © TZWorks LLC

Webpage: http://www.tzworks.com/prototype_page.php?proto_id=21

Contact Information: info@tzworks.com

1 Introduction

wisp is a Windows parser that targets NTFS index type attributes. The NTFS index attribute points to one or more *INDX* records. These records contain index entries that are used to account for each item in a directory. An index item represents either a file or a subdirectory and includes enough metadata to contain the name, modified/access/MFT changed/birth (*MACB*) timestamps, size (if it is a file vice subdirectory), as well as MFT entry numbers of the item and its parent. The **wisp** tool, in its simplest form, is able to walk these structures, read the metadata, and report which index entries are present.

As a directory's contents are changed, the number of valid index entries grows or shrinks, as appropriate. As more directory entries are added, eventually it will exceed the existing *INDX* record allocation space. At this point, the operating system will allocate an additional *INDX* record in the size of 0x1000 byte chunk. Conversely, when entries are removed from the directory, the *INDX* record space is not necessarily deallocated. Thus, anytime the number of index entries shrinks, the invalid ones potentially can be harvested from the slack space. The slack space is defined to be the allocated, but unused, space. By comparing both the valid entries and those still in the slack space, one can make some inferences about whether a file (or subdirectory) was present in the past.

A good tutorial on harvesting index entries from *INDX* slack space can be found on Willi Ballenthin's webpage [4] and his *DFIRonline* presentation [5].

wisp uses the NTFS and index attribute parsing engine that is used in the **ntfswalk** tool [6] available from the TZWorks LLC website. Currently, there are compiled versions for Windows, Linux and Mac OS-X.

2 *INDX* Attributes (Format/Internals)

NTFS uses two types of index *attributes* to store directory items: (a) *INDEX_ROOT* and the (b) *INDEX_ALLOCATION*. The former is meant for a small number of index items, since it is resident within the file record of the MFT entry. The term *resident* means the data is stored within the file record's space, which is limited to a fixed size. Since the file record needs this fixed space to store its other attributes as well, there is only enough space to store only a few index entries, which are reserved for the root directory indexes. These leads to the latter attribute which is classified as *non-resident*. The *non-resident* quality of the *INDEX_ALLOCATION* attribute allows any associated data to exist as one or more separate '*cluster runs*' within the NTFS volume. Consequently, the data associated with this

attribute can grow to whatever size is needed to account for all the index items identified in a directory. There is a good explanation of these two specific attributes in Brian Carrier's book on *File System Forensic Analysis* [3] as well as other sources online [1, 2]. While **wisp** parses both attributes identified above, it is the latter attribute that contains sufficient slack space for **wisp** to analyze older index entries.

When parsing index attributes (or any NTFS attribute for that matter), it is useful to take each attribute apart at the most basic level and extract all the fields to find the best way to analyze it. For example, when looking at any directory on the file system that contains more than a few index entries, one should be able to see both the *INDEX_ROOT* and *INDEX_ALLOCATION* attributes. Below is a snapshot of the attributes of interest for the some arbitrary directory. For clarity, all the attributes have been trimmed out with the exception of the *INDEX_ROOT* and *INDEX_ALLOCATION* attributes, since this is what **wisp** will analyze. One can see that the *INDEX_ROOT* entry, which is *resident*, only has 176 (0xb0) bytes allocated for data, while the *INDEX_ALLOCATION* attribute has 12288 (0x3000) bytes allocated for data. Both attributes must be parsed to give one a complete picture of all the index entries associated within a directory. Therefore, **wisp** will first pull out the data from the *resident* attribute and parse it. Next, it will identify the cluster run(s) for the *non-resident* data, read the clusters, fix-up the *INDX* records, and extract each index entry found.

Looking at the NTFS Index type attributes for directory entries

Index Root

Index Allocation

These are the ones to target slack data from

Upon examining *non-resident* *INDX* records, one can immediately see each *INDEX_ALLOCATION* record starts with the signature '*INDX*' followed by some metadata. Some of the metadata allows one to verify the integrity of the *INDX* record as well as fix-up the record on sector boundaries. Also included in the header info is a *DIRECTORY_INDEX* record to identify the space used to store all the index entries. For details on the structure makeup, see ref [1, 2, or 3]. After the header information, the index entries

follow. These are parsed by **wisp** to enumerate both valid index entries and those which are in slack space.

| | | | |
|----------|---|------|---------------|
| 00000000 | 49 4e 44 58 28 00 09 00 52 3c 26 09 00 00 00 | INDX | W <6 |
| 00000010 | 00 00 00 00 00 00 00 00 00 00 20 07 00 00 | | |
| 00000020 | e8 0f 00 00 00 00 00 00 00 00 14 00 44 00 | | |
| 00000030 | 14 00 00 00 00 00 00 00 00 00 00 00 00 00 | | |
| 00000040 | 5d 11 01 00 00 00 0e 00 60 00 50 00 00 00 00 | | P |
| 00000050 | b1 8c 01 00 00 00 14 00 2f b4 d4 19 48 2d cd 01 | | |
| 00000060 | 9a 21 ab 84 48 2d cd 01 9a 21 ab 84 48 2d cd 01 | | |
| 00000070 | ee e6 f6 83 48 2d cd 01 00 00 1c 02 00 00 00 00 | | |
| 00000080 | 00 00 1c 02 00 00 00 00 20 00 00 00 00 00 00 | | |
| 00000090 | 07 03 75 00 73 00 6e 00 2e 00 62 00 69 00 6e 00 | | u.s.n. b.i.n. |
| 000000a0 | 6c bd 03 00 00 00 8c 00 78 00 68 00 00 00 00 00 | | x.h. |
| 000000b0 | b1 8c 01 00 00 00 14 00 19 cf 75 33 1f 44 cd 01 | | u3 D |
| 000000c0 | 7a 30 78 33 1f 44 cd 01 fc 7e 1d 43 be 4f cd 01 | | 20x |
| 000000d0 | 19 cf 75 33 1f 44 cd 01 00 00 98 00 00 00 00 00 | | u |
| 000000e0 | 00 00 98 00 00 00 00 00 20 00 00 00 00 00 00 | | |

3 How to Use *wisp*

While the *wisp* tool doesn't require one to run with administrator privileges, without doing so will restrict one to only looking at off-line '*dd*' images. Therefore, to perform *live* processing of volumes, one needs to launch the command prompt with administrator privileges.

One can display the menu options by typing in the executable name without parameters. A screen shot of the menu is shown below.



```
Administrator: Windows PowerShell

Usage

wisp [source of NTFS volume] [item to analyze] [output options]

[source of NTFS volume options]
-path <dir>                        = analyze directory
-image <file> [-offset <vol>] [-path <dir>] = src is dd image
-partition <drv letter> [-path <dir>]
-drivenum <#> [-offset <vol>] [-path <dir>] = *** phys drive#
-vmdk "disk1 | ..." [-path <dir>]      = VMWare disk(s)
-indxfile <indx datafile>                = INDX file
-vss <num>                               = *** Vol Shadow

[suboption for above src opts - cannot be used with (-indxfile)]
-mft <dir inode to analyze>

Basic options
-csv                = output in comma separated value format
-csvl2t             = log2timeline output
-bodyfile           = sleuthkit output
-valid              = output only valid entries (default)

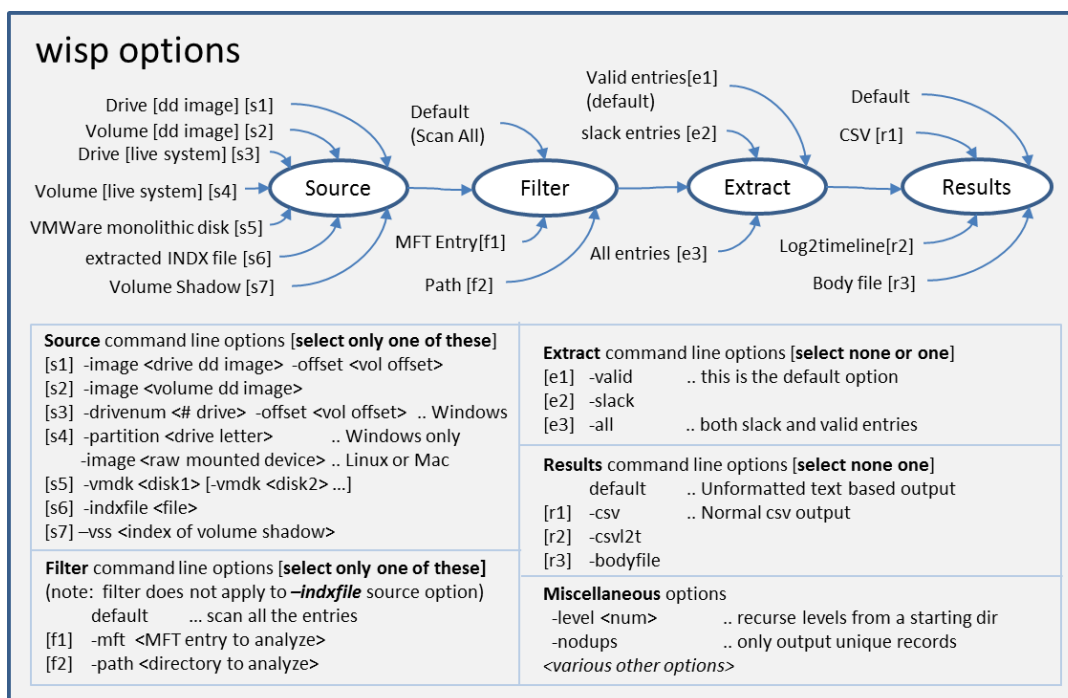
Additional options
-nodups             = output w/ no duplicate records
-slack              = output only slack entries
-all               = output both valid and slack entries
-level <num>        = recurse # levels [default is 0 levels]
-base10             = output in base10 vice std::hex
-username <name>    = output will contain this username
-hostname <name>     = output will contain this hostname
-dateformat mm/dd/yyyy = "yyyy-mm-dd" is the default
-timeformat hh:mm:ss  = "hh:mm:ss.xxx" is the default
-pair_datetime       = *** combine date/time into 1 field for csv
-no_whitespace       = remove whitespace between csv delimiter
-quiet               = don't display status during run
-csv_separator "|"    = use a pipe char for csv separator
-hexdump              = *** incl std::hex dump [not for -csv options]
```

From the available options, one can process NTFS *INDX* records with a handful of '*use-cases*'. Specifically, *wisp* allows processing from any of these sources: (a) *live* volume, (b) '*dd*' type image, (c) *VMWare* volume, or (d) separately extracted *INDX* type record. After selecting the source of the data, one can either: (a) process a single directory on the file system, (b) recursively process the subdirectories to some specified level, or (c) process all the index entries in an entire volume. Processing every directory in the entire volume is not explicitly shown in the above menu since it is the default option.

If one only wants a certain type of index entry, one can select: (a) just show valid index entries, (b) just show index entries in the slack space, or (c) both. For default output, the data is represented unstructured text. If parsable output is desired (or something that can be displayed in a spreadsheet

application), one can select from 3 options that allow for structured output (CSV, *log2timeline* CSV [8], or *SleuthKit body-file* [9]). The other useful option is the '*no duplicates*' choice to minimize any redundancy in the output. There is a discussion of why one might want to use the '*no duplicates*' option in a later section.

Since there are a number of possible combinations of options, the figure below shows the possible choices and where they apply in a logical processing flow. The comments in the figure annotate any restrictions for a particular option.



3.1 Parsing a Live Volume

To parse *INDX* entries from a live NTFS volume (or partition), one has two choices: (a) specify the volume directly by using the **-partition <drive letter>** option or (b) specify the drive number and volume offset by using the **-drivenum <num> -offset <volume offset>** option. Either choice accomplishes the same task. The first choice is more straightforward and easier to use. The second choice, while more complex, allows one to target hidden NTFS partitions that do not have a drive letter.

The next step is to decide what you want to target. The choices are: (a) a specific directory on the file system (specified by either the **-mft** or **-path** options) or (b) a collection of subdirectories within a directory (how deep you wish to go is specified by the **-level** option). A couple examples are shown below:

```
wisp -path c:\$Recycle.Bin -level 2 -all -csv > results1.csv
```

```
wisp -partition c -all -csv > results2.csv
```

```
wisp -drivenum 0 -offset 0x100000 -all -csv > results3.csv
```


The first example targets the hidden directory of `c:\$Recycle.Bin`, and the **-level 2** switch tells *wisp* to include any subdirectory in the analysis, up to 2 levels deep. The **-all** switch means both *valid* and *invalid* (slack) entries will be included in the output. Finally, the output is redirected to a file and the format is CSV.

The second example uses the same output options as the first, but now targets the 'c' partition. Since there is no **-mft** or **-path** options explicitly listed, the implication to *wisp* is we want to traverse the entire volume parsing all *INDX* records associated with the volume.

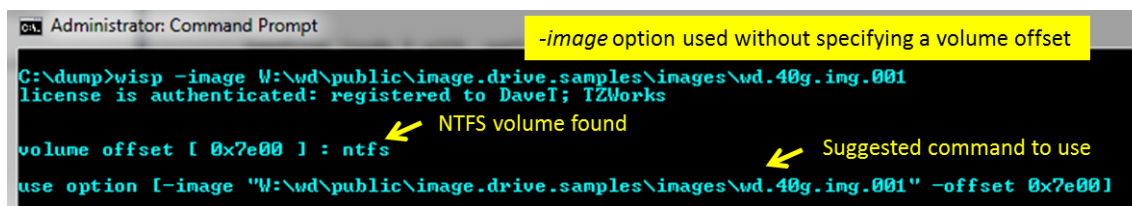
The third example uses the same output options as the second, but now targets the first physical hard drive. The hex value `0x100000` is specified as the offset to the volume (or partition) we wish to analyze. For this example, this happens to be the hidden partition created during a Windows 7 installation. Since there is no **-mft** or **-path** options explicitly listed, the implication to *wisp* is we want to traverse the entire volume parsing all *INDX* records associated with the volume..

3.2 Parsing an Image File off-line

To process an image that has been already acquired and is in the 'dd' format, one uses the **-image** switch. This option can be used in two flavors. If the image is of an entire drive, then one needs to explicitly specify the *offset* of the location of the volume you wish to target. On the other hand, if the image is only of a volume, then you do not need to specify the offset of the volume (since it is presumed to be at offset 0).

For the first case, where an offset needs to be explicitly specified, *wisp* will help the user in locating where the NTFS volume offsets are. If one just issues the **-image** command without the offset, and there is *not* a NTFS volume at offset 0 (eg. second case mentioned above), *wisp* will proceed to look at the master boot record contained in the image, determine where the NTFS partitions are, and report them to the user. This behavior was meant to be an aid to the user so that one does not need to resort to other tools to determine where the offsets for the NTFS volumes are in an image. Below is a screenshot what is displayed to the user in this situation.

Shown in the screenshot is a **-image** command that is issued without the offset. *wisp* detects that the image is of an entire drive vice of a volume and locates one NTFS volume at offset `0x7e00` hex. *wisp* then reports to the user a suggested syntax (of the command, if re-issued) to process this volume.



```
Administrator: Command Prompt
C:\>wisp -image W:\wd\public\image.drive.samples\images\wd.40g.img.001
license is authenticated: registered to DaveI; TZWorks
volume offset [ 0x7e00 ] : ntfs
use option [-image "W:\wd\public\image.drive.samples\images\wd.40g.img.001" -offset 0x7e00]
```


Another nuance with using images as the source, is that when specifying a path to a directory within the image to analyze one should use the **-path** option. Since the image is not mounted as a drive, one really should not associate it with a drive letter when specifying the path. If one does do this, **wisp** will ignore the drive letter and proceed to try to find the path starting at the root directory which is at MFT entry 5 for NTFS volumes.

Below are two examples of processing 'dd' type images: (a) the first analyzes an entire volume at drive offset 0x100000 hex and (b) the second analyzes an image of a volume starting at the path "Users".

```
wisp -image c:\dump\my_image.dd -offset 0x100000 -all -csv > results1.csv
```

```
wisp -image c:\dump\vol_image.dd -path "Users" -level 5 -all -csv > results2.csv
```

While the first example traverses the entire volume, the second starts at the "Users" directory, and recursively processes the subdirectories up to 5 levels deep. Notice the second example does not specify an offset, since the image is of a volume (meaning the volume starts at offset 0) while the first is an image of a drive and the first NTFS volume starts at offset 0x100000 hex.

Both examples extract *valid* and *invalid* index entries as well as redirect their output to a file using CSV formatting.

3.3 Handling Directories in Volume Shadows

For starters, to access Volume Shadow copies, one needs to be running with administrator privileges. Also, Volume Shadow copies, as is discussed here, only applies to Windows Vista, Win7, Win8 and beyond. It does not apply to Windows XP.

To tell **wisp** to look at a Volume Shadow, one needs to use the **-vss <index of volume shadow>** option. This then points **wisp** at the appropriate Volume Shadow and start processing the desired directory.

Below are 2 examples. The first will traverse the Users directory to level of 4 deep for the Volume Shadow copy specified by index 1. The second will traverse all the directories in Volume Shadow copy specified by index 2.

```
wisp -vss 1 -path \Users -level 4 -csv > out.csv
```

```
wisp -vss 2 -csv > out.csv
```

To determine which indexes are available from the various Volume Shadows, one can use the Windows built-in utility **vssadmin**, as follows:

```
vssadmin list shadows
```

To filter some of the extraneous detail, type

```
vssadmin list shadows / find /i "volume"
```

While the amount of data can be voluminous from that above command, the keywords one needs to look for are names that look like this:

```
Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1
```

Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy2

...

From the above, notice the number after the word *HarddiskvolumeShadowCopy*. It is this number that is passed as an argument to the **-vss** option.

3.4 Parsing a NTFS Volume Mounted on Linux or Mac OS-X

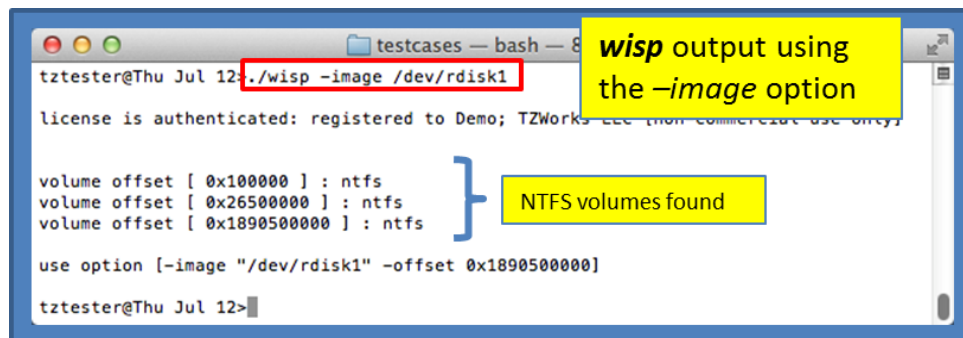
Sometimes you do not have a *'dd'* image of the volume or drive, but instead have the physical hard drive available you wish to analyze. If you are running **wisp** in Windows, then follow the guidelines in the earlier section for parsing a *live* volume. However, if you are running **wisp** in Linux or Mac OS-X, you should also be able to mount the target drive as well. Once it is successfully mounted, one uses the **-image <device name of drive or volume> -offset <offset to desired volume, if a drive>** option to access the appropriate NTFS volume. Below is an example of how to do this using a Mac box.

Assuming one has the proper setup with write blocker and hard drive shuttle, after connecting the Windows drive to the Mac, one can issue the **diskutil list** command to enumerate all the drives and volumes mounted on the machine. For the configuration under test, the results showed the following:

```
tzttester@Thu Jul 12>diskutil list
/dev/disk0
#:                                TYPE NAME              SIZE       IDENTIFIER
0:    GUID_partition_scheme      *320.1 GB   disk0
1:      EFI                      209.7 MB   disk0s1
2:      Apple_HFS Lion            150.8 GB   disk0s2
3:      Apple_Boot Recovery HD    650.0 MB   disk0s3
4:      Apple_HFS SnowLeopard     168.3 GB   disk0s4

/dev/disk1
#:                                TYPE NAME              SIZE       IDENTIFIER
0:    FDisk_partition_scheme      *160.0 GB   disk1
1:      Windows_NTFS System Reserved 641.7 MB   disk1s1
2:      Windows_NTFS               104.9 GB   disk1s2
3:      Windows_NTFS Workarea       54.5 GB    disk1s3
```

The second disk (designated as */dev/disk1*) is the external disk that was plugged in, and it consisted of 3 NTFS partitions (*disk1s1*, *disk1s2*, *disk1s3*). Once the device for the disk has been identified, one can reference the partitions directly with their respective identifiers. If for some reason, the **diskutil** does not identify the partitions due to disk corruption, one can also reference the disk directly and explicitly give the offset of the desired partition one wishes to operate on. Below is how one could use **wisp** to help identify the partition offsets.



```
testcases -- bash -- 8
tztester@Thu Jul 12: ./wisp -image /dev/rdisk1
license is authenticated: registered to Demo; TZWorks LLC (non-commercial use only)

volume offset [ 0x100000 ] : ntfs
volume offset [ 0x26500000 ] : ntfs
volume offset [ 0x1890500000 ] : ntfs
} NTFS volumes found

use option [-image "/dev/rdisk1" -offset 0x1890500000]
tztester@Thu Jul 12>
```

Notice, the command that was issued above was just the device name of the disk and **wisp** looked at the *MBR* to determine where NTFS partitions were located relative to the start of the disk. Also notice, the notation of `/dev/rdisk#` is used. The '*r*' (which is unique to Mac, in this case) is used to specify we want to access the drive as '*raw*' I/O as opposed to buffered I/O. The buffering is nice for normal reads/writes, but it is much slower when traversing in chunks aligned on sector boundaries.

Based on the above discussion, below are the possible options to access the *first* NTFS volume.

```
sudo wisp -image /dev/rdisk1s1 -all -csv -nodups > out.csv
```

```
sudo wisp -image /dev/rdisk1 -offset 0x100000 -all -csv -nodups > out.csv
```

Notice the '*sudo*' in front of the **wisp** command. This will allow **wisp** to run with administrative privileges to access the raw drive. Linux is similar to Mac, but instead of using the **diskutil** tool, one would use the '*df*' tool to enumerate the mounted devices.

3.5 Parsing a VMWare Volume

Occasionally it is useful to analyze a *VMWare* image containing a Windows volume, both from a forensics standpoint as well as from a testing standpoint. This option is still considered *experimental* since it has only been tested on a handful of configurations. Furthermore, this option is limited to *monolithic* type *VMWare* images versus *split* images. In *VMWare*, the term *split* image means the volume is separated into multiple files, while the term *monolithic* virtual disk is defined to be a virtual disk where everything is kept in one file. There may be more than one VMDK file in a *monolithic* architecture, where each monolithic VMDK file would represent a separate snapshot. More information about the *monolithic* virtual disk architecture can be obtained from the *VMWare* website [10].

When working with virtual machines, the capability to handle snapshot images is important. Thus, if processing a *VMWare* snapshot, one needs to include the desired snapshot/image as well as its inheritance chain.

wisp can handle multiple VMDK files to accommodate a snapshot and its descendants, by separating multiple filenames with a pipe delimiter and enclosing the expression in double quotes. In this case, each filename represents a segment in the inheritance chain of VMDK files (eg. **-vmdk "<VMWare NTFS virtual disk-1> | .. | <VMWare NTFS virtual disk-x>"**). To aid the user in figuring out exactly the chain

of descendant images, **wisp** can take any VMDK file (presumably the VMDK of the snapshot one wishes to analyze) and determine what the descendant chain is. Finally, **wisp** will suggest a chain to use.

Aside from the VMDK inheritance chain, everything else is the same when using this option to that of normal 'dd' type images discussed in the previous section.

3.6 **wisp** Output

3.6.1 Which Output option yields the most data

The two output options that give all the metadata available are the default (unstructured) output and the CSV (**-csv**) output. The other two output options (**-csv12t** and **-bodyfile**) are geared toward generating cross-artifact timelines. As a result, they are more restrictive in the output fields, and therefore, the metadata that is parsable from these options have some limitations. Specifically, **wisp** makes use of a couple of free-form fields in these last two output options to try to inject as much useful data as possible, but it makes the data in these fields unstructured, and therefore, difficult to parse if trying to post process the results. Therefore, the best option for metadata that needs to be parsed from **wisp** output comes from the **-csv** option.

3.6.2 Two Categories of Slack entries

The slack entries in the output have comments associated with them. The comments come in 2 categories: (a) entries that have not been deleted and (b) those that have been deleted. These categories are best shown with an example. Using the default output option, 2 snapshots are shown below.

The first snapshot shows a valid index entry as well as an invalid index entry. Both index entries point to the same file (one can tell this since the MFT entry number and sequence numbers match up). The difference, aside from the fact one is in slack space, is some of the *MACB* timestamps are different as well as the size of the file. **wisp** annotates the modification with a comment, denoted by *[m.c.]* and *[size]* in the snapshot below. The *[m.c.]* translates to the *modify* timestamp and 'MFT change' timestamps, respectively, as being different than the valid entry. The *[size]* notation just means the size of the file has changed. From this, one can get some past data on a file that has gone through some revisions.

Slack entries can provide evidence of file modification by comparing *like* entries that are in both valid and invalid category

```

Name: c:\users\tzlabs\desktop\todo.txt
MFT Entry: 0x00000003930e [234254]
MFT Sequence: 0x000c [12]
Parent Entry: 0x00000000004c [76]
Parent Sequence: 0x0003 [3]
Entry: valid
Modify time [utc]: 129860511243043130 [07/06/12 12:25:24.304]
Access time [utc]: 129689668274077974 [12/21/11 18:47:07.407]
MFTChanged time [utc]: 129860511243043130 [07/06/12 12:25:24.304]
Create time [utc]: 129689668273867947 [12/21/11 18:47:07.386]
Size allocated: 0x00002000 [8192]
Size used: 0x000014d5 [5333]
Type entry: 0x00000020 [ARCHIVE]
Source type: indx alloc

Name: c:\users\tzlabs\desktop\todo.txt
MFT Entry: 0x00000003930e [234254]
MFT Sequence: 0x000c [12]
Parent Entry: 0x00000000004c [76]
Parent Sequence: 0x0003 [3]
Entry: slack, modified [m.c.] [size]
Modify time [utc]: 129813890214116606 [05/13/12 13:23:41.411]
Access time [utc]: 129689668274077974 [12/21/11 18:47:07.407]
MFTChanged time [utc]: 129813890214116606 [05/13/12 13:23:41.411]
Create time [utc]: 129689668273867947 [12/21/11 18:47:07.386]
Size allocated: 0x00002000 [8192]
Size used: 0x000010e1 [4321]
Type entry: 0x00000020 [ARCHIVE]
Source type: indx alloc

```

Current (valid) entry

Slack entry

For the second case, shown in the below snapshot, **wisp** just annotates the slack entry as *deleted*. The term *deleted* here is only accurate in the sense that this index entry is no longer part of the directory containing the *INDX* record(s). Whether the item was *moved* to another subdirectory or actually *deleted* is unknown from the data presented here.

Slack entries can provide evidence of prior presence

```

Name: c:\users\tzlabs\desktop\uni.config.txt
MFT Entry: 0x00000009d80a [645130]
MFT Sequence: 0x0002 [2]
Parent Entry: 0x00000000004c [76]
Parent Sequence: 0x0003 [3]
Entry: slack, deleted
Modify time [utc]: 129814844549650112 [05/14/12 15:54:14.965]
Access time [utc]: 129814844994968634 [05/14/12 15:54:59.496]
MFTChanged time [utc]: 129814845076783426 [05/14/12 15:55:07.678]
Create time [utc]: 129814844994968634 [05/14/12 15:54:59.496]
Size allocated: 0x00000180 [384]
Size used: 0x0000017b [379]
Type entry: 0x00000020 [ARCHIVE]
Source type: indx alloc

```

Slack entry

Finally, looking at the data available in an index entry one can see the *MACB* timestamps and size, as well as the MFT entry metadata, flags and source of information. The *MACB* timestamps should match the standard information *MACB* timestamps of the file/subdirectory.

3.6.3 Corruption of the Index Entry

In cases where the *slack* data is obviously corrupted, **wisp** will either leave that field blank if using CSV output or annotate the word *<corrupted>* if using the default output.

3.6.4 Looking at the Raw Data for Slack Entries

Occasionally, the examiner wants to see the raw data for an entry. This can be done by using the **-hexdump** switch and using the long format option (eg. non-CSV). The output will show each entry parsed annotated with a hex dump of the raw data. Below is an example of what this output looks like.

```
wisp - full ver: 0.22; Copyright (c) TZWorks LLC
License is authenticated for business use and registered to Dave T; TZWorks
run time: 04/02/2014 18:29:44 [UTC]
cmdline: wisp64 -partition c -path c:\dump -hexdump -slack

Name: c:\dump\lp.dbg.out.txt
MFT Entry: 0x000000014a3c [84540]
MFT Sequence: 0x0017 [23]
Parent Entry: 0x00000000e5b7 [58807]
Parent Sequence: 0x0003 [3]
Entry: slack, exists [dup slack entry]
Modify time: 03/26/2014 22:03:52.395 [UTC]
Access time: 03/26/2014 22:03:52.394 [UTC]
MFTChanged time: 03/26/2014 22:04:02.751 [UTC]
Create time: 03/26/2014 22:03:52.394 [UTC]
Size allocated: 0x00000070 [112]
Size used: 0x0000006c [108]
Type entry: 0x00000020 [ARCHIVE]
Source type: indx_alloc

0000 08a0: 3c 4a 01 00 00 00 17 00 70 00 5e 00 00 00 00 00 <J.....p.^.....
0000 08b0: b7 e5 00 00 00 00 03 00 ca 54 0b 46 3f 49 cf 01 .....T.F?I...
0000 08c0: dc 7b 0b 46 3f 49 cf 01 e2 a7 37 4c 3f 49 cf 01 .{.F?I....7L?I..
0000 08d0: ca 54 0b 46 3f 49 cf 01 70 00 00 00 00 00 00 00 .T.F?I..p.....
0000 08e0: 6c 00 00 00 00 00 00 00 20 00 00 00 00 00 00 00 l.....
0000 08f0: 0e 01 6c 00 70 00 2e 00 64 00 62 00 67 00 2e 00 ..l.p...d.b.g...
0000 0900: 6f 00 75 00 74 00 2e 00 74 00 78 00 74 00 00 00 o.u.t...t.x.t.

Name: c:\dump\sans.sift.tar.gz
MFT Entry: 0x000000000590 [1424]
MFT Sequence: 0x0017 [23]
Parent Entry: 0x00000000e5b7 [58807]
Parent Sequence: 0x0003 [3]
Entry: slack, exists [dup slack entry]
Modify time: 03/27/2014 12:36:41.681 [UTC]
Access time: 03/27/2014 12:37:04.735 [UTC]
MFTChanged time: 03/27/2014 12:37:08.887 [UTC]
Create time: 03/27/2014 12:37:04.735 [UTC]
Size allocated: 0x00afe000 [11526144]
Size used: 0x00afd2e8 [11522792]
Type entry: 0x00000020 [ARCHIVE]
Source type: indx_alloc

0000 0ce0: 90 05 00 00 00 00 17 00 78 00 62 00 00 00 00 00 .....x.b.....
0000 0cf0: b7 e5 00 00 00 00 03 00 e9 10 50 42 b9 49 cf 01 .....PB.I...
0000 0d00: 16 55 92 34 b9 49 cf 01 7a 9d c9 44 b9 49 cf 01 .U.4.I..z..D.I..
0000 0d10: e9 10 50 42 b9 49 cf 01 00 e0 af 00 00 00 00 00 00 ..PB.I.....
0000 0d20: e8 d2 af 00 00 00 00 00 20 00 00 00 00 00 00 00 .....
0000 0d30: 10 01 73 00 61 00 6e 00 73 00 2e 00 73 00 69 00 ..s.a.n.s...s.i.
0000 0d40: 66 00 74 00 2e 00 74 00 61 00 72 00 2e 00 67 00 f.t...t.a.r...g.
0000 0d50: 7a 00 z.
```


3.7 Extracting Clusters Associated with a Deleted Entry

At this point in the analysis, one may want to go deeper and try to find if the deleted file is still available by trying to find the *'cluster run'* data associated with the MFT entry. Since the *INDX* records do not have any *cluster run* data associated with an index entry, one would need to use the MFT entry specified and then use some other tool to read the file record associated with that MFT entry. One could extract the data either from the local volume or from the volume shadow copy store. If pulling from the local volume, one can use the **ntfscopy** utility [7] from TZWorks. This tool will allow one to (a) input a volume (live or off-line), (b) specify the desired MFT entry to copy from and (c) output the extracted data associated with the MFT's *cluster run* as well as the metadata associated with that MFT entry. Below is an example of doing this with **ntfscopy** using the MFT entry number 645130, which is for the *slack* entry shown above.

```
ntfscopy -mft 645130 -dst c:\dump\645130.bin -partition c: -meta
```

For details on the **ntfscopy** syntax, refer to the **ntfscopy** readme file [7]. Briefly, the **-mft** option allows one to specify a source MFT entry to copy from. The **-meta** option says to create a separate file (in addition to the copied file) that contains the metadata information about the specified MFT entry. The metadata file will be created with the same name as the destination file with the appended suffix **meta.txt**. Included in the metadata file are many of the NTFS attributes of the target source file (or MFT entry). This includes, amongst other things, the *cluster run* and *MACB* timestamps. From the metadata one can see if the MFT sequence number is the same or not (which would be the indication whether the MFT record was assigned to another file or not).

3.8 Eliminating the Duplicates

For those *INDX* records that have many slack entries, it is not uncommon for there to be quite a few duplicate entries that are parsed and displayed in the output. Duplicate here means the filename and *MACB* timestamps are the same, however the location of the entry in the *INDX* record is different. For every unique location in the *INDX* record, **wisp** will happily parse the index entry and report its findings to the investigator. This can be quite annoying when some entries have more than a few duplicates and one is trying to wade through a lot of data; especially when carving out entries from *slack* data on all the directories in an entire volume.

To get rid of duplicates, one can invoke the **-nodups** switch. This tells **wisp** to analyze the data extracted and only report one instance of the entry. One thing to be aware of when using this option, is that **wisp** will internally always analyze *valid* and *slack* entries independent of what the user selects as input options. After all the data is extracted, **wisp** will start deciding if there are duplicate entries or not. It does this by looking at comparing all *slack* entries with *valid* entries to see if there is a duplicate, and if not, they are compared to any *slack* entries that have been marked as non-dups. What this means is, if

one runs **wisp** and only wants non-duplicate *slack* entries, some slack entries will not be reported if there are valid entries present that are the same.

4 Known Issues

For CSV (comma separated values) output, there are restrictions in the characters that are outputted. Since commas are used as a separator, any data that had comma in its name are changed to spaces. For the default (non-csv) output no changes are made to the data.

5 Available Options

The options labeled as 'Extra' require a separate license for them to be unlocked.

| Option | Description |
|-------------------|--|
| -path | Analyze the index entries given a path. The syntax is: -path <directory to analyze> . |
| -image | Extract artifacts from a volume specified by an image and volume offset. The syntax is -image <filename> -offset <volume offset> |
| -partition | Extract artifacts from a mounted Windows volume. The syntax is -partition <drive letter> . |
| -drivenum | Extract artifacts from a mounted disk specified by a drive number and volume offset. The syntax is -drivenum <#> -offset <volume offset> |
| -vmdk | Extract artifacts from a VMWare monolithic NTFS formatted volume. The syntax is -vmdk <disk name> . For a collection of VMWare disks that include snapshots, one can use the following syntax: -vmdk "disk1 disk2 ..." |
| -vss | Experimental. Extract INDX data from Volume Shadow. The syntax is -vss <index number of shadow copy> . Only applies to Windows Vista, Win7, Win8 and beyond. Does not apply to Windows XP. |
| -csv | Outputs the data fields delimited by commas. Since filenames can have commas, to ensure the fields are uniquely separated, any commas in the filenames get converted to spaces. |

| | |
|-----------------------|--|
| -csvl2t | Outputs the data fields in accordance with the log2timeline format. |
| -bodyfile | Outputs the data fields in accordance with the 'body-file' version3 specified in the SleuthKit. The date/timestamp outputted to the body-file is in terms of UTC. So if using the body-file in conjunction with the mactime.pl utility, one needs to set the environment variable TZ=UTC. |
| -base10 | Ensure all size/address output is displayed in base-10 format vice hexadecimal format. Default is hexadecimal format. |
| -mft | Extract index metadata given an inode. The syntax is: -mft <MFT entry to analyze> . |
| -indxfile | Process INDX data that may be been extracted from another tool. The syntax is: -indxfile <datafile name> ; |
| -valid | Extract the metadata from the valid index entries only. |
| -slack | Extract the metadata from the slack index entries only. |
| -all | Pull every index entry found. This includes both valid and invalid (slack) index entries. |
| -nodups | Output unique entries only. Duplicate entries are flagged if the name and MACB timestamps are the same. |
| -username | Option is used to populate the output records with a specified username. The syntax is -username <name to use> . |
| -hostname | Option is used to populate the output records with a specified hostname. The syntax is -hostname <name to use> . |
| -no_whitespace | Used in conjunction with -csv option to remove any whitespace between the field value and the CSV separator. |
| -csv_separator | Used in conjunction with the -csv option to change the CSV separator from the default comma to something else. Syntax is -csv_separator "/" to change the CSV separator to the pipe character. To use the tab as a separator, one can use the -csv_separator "tab" OR -csv_separator "\t" options. |

| | |
|-----------------------|--|
| -dateformat | Output the date using the specified format. Default behavior is -dateformat "yyyy-mm-dd" . Using this option allows one to adjust the format to mm/dd/yy, dd/mm/yy, etc. The restriction with this option is the forward slash (/) or dash (-) symbol needs to separate month, day and year and the month is in digit (1-12) form versus abbreviated name form. |
| -timeformat | Output the time using the specified format. Default behavior is -timeformat "hh:mm:ss.xxx" One can adjust the format to microseconds, via "hh:mm:ss.xxxxxx" or nanoseconds, via "hh:mm:ss.xxxxxxxxxx" , or no fractional seconds, via "hh:mm:ss" . The restrictions with this option is a colon (:) symbol needs to separate hours, minutes and seconds, a period (.) symbol needs to separate the seconds and fractional seconds, and the repeating symbol 'x' is used to represent number of fractional seconds. (Note: the fractional seconds applies only to those time formats that have the appropriate precision available. The Windows internal filetime has, for example, 100 nsec unit precision available. The DOS time format and the UNIX 'time_t' format, however, have no fractional seconds). Some of the times represented by this tool may use a time format without fractional seconds, and therefore, will not show a greater precision beyond seconds when using this option. |
| -pair_datetime | Output the date/time as 1 field vice 2 for csv option |
| -quiet | This option suppresses status output during processing. |
| -hexdump | This option dovetails hex output after parsed data. Useful for verification of parsed data. Only available for unstructured default output (eg. not for -csv , -csvl2t or -bodyfile outputs). |
| -utf8_bom | All output is in Unicode UTF-8 format. If desired, one can prefix an UTF-8 byte order mark to the CSV output using this option. |

6 Authentication and the License File

This tool has authentication built into the binary. The primary authentication mechanism is the digital X509 code signing certificate embedded into the binary (Windows and macOS).

The other mechanism is the runtime authentication, which applies to all the versions of the tools (Windows, Linux and macOS). The runtime authentication ensures that the tool has a valid license. The

license needs to be in the same directory of the tool for it to authenticate. Furthermore, any modification to the license, either to its name or contents, will invalidate the license.

6.1 *Limited versus Demo versus Full* in the tool's Output Banner

The tools from *TZWorks* will output header information about the tool's version and whether it is running in *limited*, *demo* or *full* mode. This is directly related to what version of a license the tool authenticates with. The *limited* and *demo* keywords indicates some functionality of the tool is not available, and the *full* keyword indicates all the functionality is available. The lacking functionality in the *limited* or *demo* versions may mean one or all of the following: (a) certain options may not be available, (b) certain data may not be outputted in the parsed results, and (c) the license has a finite lifetime before expiring.

7 References

1. <http://www.ntfs.com> website.
2. <http://en.wikipedia.org/wiki/NTFS> website
3. Brian Carrier's book, File System Forensic Analysis, sections on NTFS
4. Willi Ballenthin article on [NTFS INDX parsing](#).
5. Getting to know your NTFS INDX Records presented May 3, 2012 on [DFIRonline](#) by Willi Ballenthin
6. *ntfswalk* tool. http://tzworks.com/prototype_page.php?proto_id=12, TZWorks LLC.
7. *ntfscopy* tool. http://tzworks.com/prototype_page.php?proto_id=9, TZWorks LLC.
8. *SleuthKit* [Body-file](#) format, <http://wiki.sleuthkit.org>
9. Log2timeline CSV format, <http://log2timeline.net/>
10. VMWare Virtual Disk Format 1.1 Technical Note, <http://www.vmware.com>