TZWorks® Windows LNK Parser (*lp*) Users Guide

Abstract

lp is a standalone, command-line tool used to extract *SHLLINK* artifacts from Windows shortcut files. It can operate on a single *shortcut* file, a collection of *shortcut* files, or on an entire disk image. All artifacts can be outputted in one of three parsable formats for easy inclusion with other forensics artifacts. ***lp*** can also parse unallocated space to extract additional artifacts. ***lp*** runs on Windows, Linux and Mac OS-X.

www.tzworks.com

Contact Info: info@tzworks.com

*Document applies to v1.08 of **lp***

Updated: Apr 25, 2025

Table of Contents

TZWorks® LNK Parser (*lp*) Users Guide

Webpage: http://www.tzworks.com/prototype_page.php?proto_id=11
Contact Information: info@tzworks.com

1 Introduction

lp is a command line version of a Windows *SHLLINK* [2] parser that was designed to operate on *shortcut LNK* files, but can parse *SHLLINK* artifacts from files that generate *Jump Lists* as well. Originally inspired by the forensic class taken from the SANS Institute [1] back in Jan 2010, ***lp*** is a useful tool for any computer forensic toolkit.

While shortcut files can reside in just about any directory, the primary location for many *shortcut* files is: *%APPDATA%\ Microsoft\ Windows\ Recent\ <shortcut files>*, where the *%APPDATA%* is resolved to *C:\Users\<user account>\AppData\Roaming.* This is where the operating system automatically creates a shortcut based on a user double clicking on an application to launch it.

Of interest to the forensic investigator is the metadata associated with this type of file, since they offer many useful artifacts when determining activity on a computer. Some of these artifacts include:

- The path to the target file/directory it references along with modify/access/create timestamps
- The size of the target when it was last accessed.
- Serial number of the volume where the target was stored.
- Network volume share name (if applicable).
- Target attributes, such as whether it was 'read only', 'hidden', 'system', etc.
- One of the MAC addresses associated with the host computer (available when an Object ID is present).

When trying to parse out the above artifacts, one can turn to the Microsoft open specification agreement, where there is a published version of the Windows *SHLLINK* format. (ref: MS-SHLLINK [2]). From this specification, one can see many of the details needed to understand the structures of the format. Prior to Microsoft publishing this specification, there was another source describing the details of the *LNK* format. Jesse Hager's paper (ref [3]) discussed his results of reverse engineering the *LNK* internals.

The parsing engine of ***lp*** makes use of the Microsoft specification to extract much of the *shortcut* internals. Where the specification lacked details, we ended up using empirical data to help understand some of the more opaque data structure types allowing us to parse the *SHLLINK* format more fully.

2 *SHLLINK* Metadata and What *lp* Extracts

When creating tools that parse artifacts that still have unknowns associated with them, there is a balance on what data should be presented to the user and which should not. On one hand, we at *TZWorks LLC* personally like to see ***all*** the data artifacts, complete with file offsets, so we can trace each artifact in a hex editor. This allows one to hand carve the data and is very important to the reverser. However, this type of data is most likely to be too noisy for the normal user. Therefore, the version that is available commercially is a subset of options we consider useful to the general investigator. Some of these options include: (a) carving *SHLLINK* metadata from images and live volumes, (b) handling the nuances of the *Destinations* files used in the Jump Lists, and (c) additional output format options.

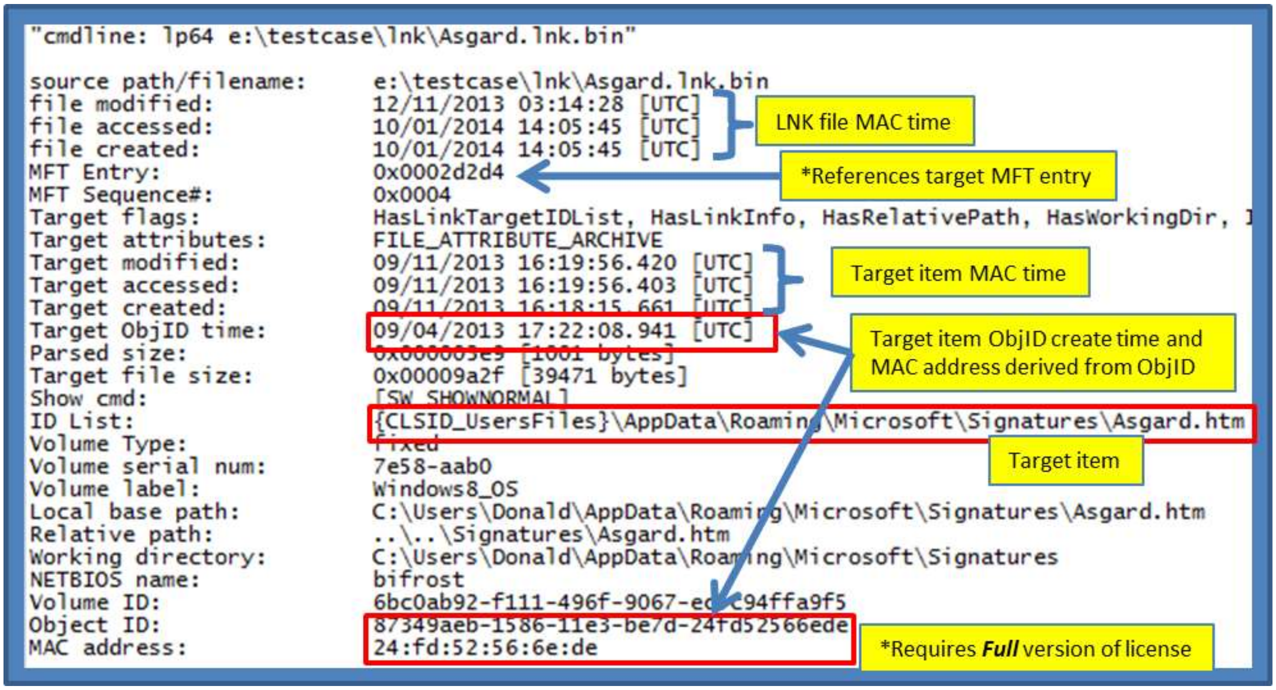
To see ***lp***'s default output, and hence some of the *SHLLINK* metadata, below are a series of snapshots that represent how various data can be embedded into the LNK file. (Note: sample LNK data was provided by Rob Lee from the SANS Institute as exemplars, which he took from the Donald Blake image used in the SANS 408 Forensics class).

2.1 Example of a more common LNK file's output

When analyzing ***lp***'s output, a number of timestamp data is shown, including *shortcut* file timestamps as well as the target (what the shortcut file points to) timestamps. The output should contain the size of the target, if it is a file versus a directory, and will contain a path to the target. As part of the *SHLLINK* specification, there is also what Microsoft calls a *TrackerDataBlock*. This is what we refer to as the object identifier (ID), since it is really the object ID of the NTFS MFT record associated with the target file or directory.

The object ID is another way to reference the target file/directory and ultimately allows the operating system a straight forward way to 'track' the target file or directory at the lower level NTFS object ID/MFT entry level. This object ID is part of the target file and moves where the target moves. In the *SHLLINK* metadata there are two object IDs: (a) one that is recorded when the shortcut is created, and (b) one that is the current one. For the most part these two object IDs are the same and will only differ in certain conditions. Internally, ***lp*** makes note of both object IDs, however, it will only display both if they are different.

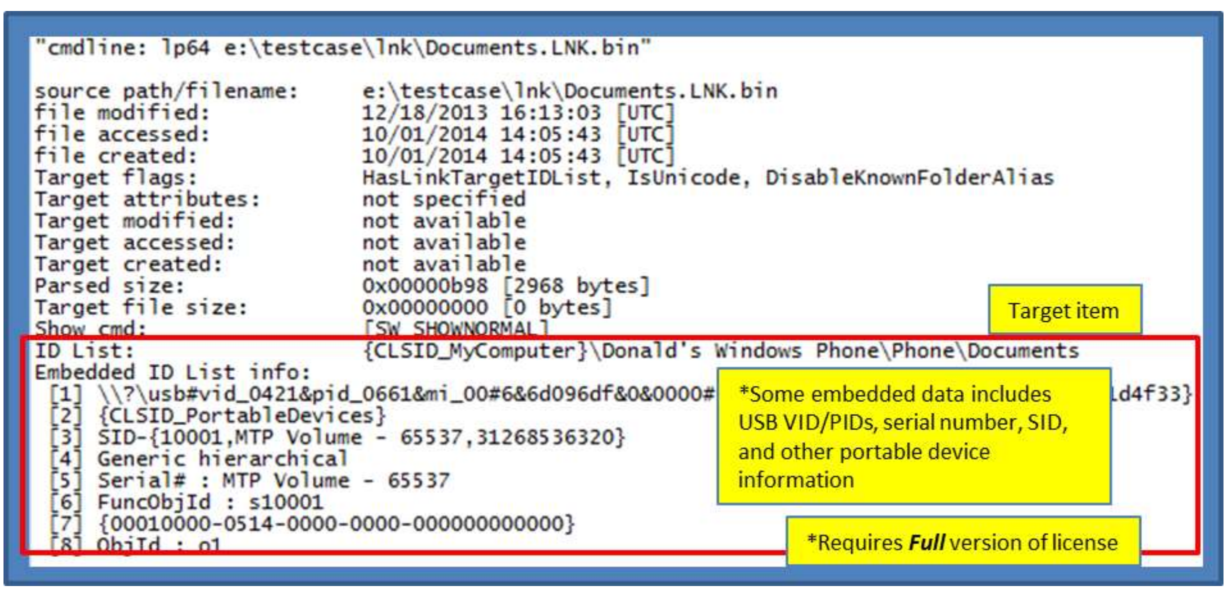
Associated with the object ID is a creation timestamp and media access control (MAC) network interface identifier that was present during the object ID creation. The format of the object ID follows the Type 1 specification outlined in RFC 4122 (Universally Unique Identifier URN Namespace) [10]. Using this specification, one can extract the object time and MAC address from the object ID itself. This means that there is not any object ID timestamp or network interface artifact *explicitly* present in the SHLLINK metadata, and any data shown in ***lp***'s output for these fields is from *implicitly* inferring it by extracting it from the object ID itself. When analyzing the below MAC network interface extracted, it identified one of our *VMWare* network interfaces and not the primary computer's network address.



```
"cmdline: lp64 e:\testcase\lnk\Asgard.lnk.bin"

source path/filename:     e:\testcase\lnk\Asgard.lnk.bin
file modified:            12/11/2013 03:14:28 [UTC]
file accessed:            10/01/2014 14:05:45 [UTC]
file created:             10/01/2014 14:05:45 [UTC]
MFT Entry:                0x0002d2d4
MFT Sequence#:            0x0004
Target flags:             HasLinkTargetIDList, HasLinkInfo, HasRelativePath, HasWorkingDir, I
Target attributes:        FILE_ATTRIBUTE_ARCHIVE
Target modified:          09/11/2013 16:19:56.420 [UTC]
Target accessed:          09/11/2013 16:19:56.403 [UTC]
Target created:           09/11/2013 16:18:15.661 [UTC]
Target ObjID time:        09/04/2013 17:22:08.941 [UTC]
Parsed size:              0x000003e9 [1001 bytes]
Target file size:         0x00009a2f [39471 bytes]
Show cmd:                 [SW_SHOWNORMAL]
ID List:                  {CLSID_UsersFiles}\AppData\Roaming\Microsoft\Signatures\Asgard.htm
Volume Type:              fixed
Volume serial num:        7e58-aab0
Volume label:             Windows8_OS
Local base path:          C:\Users\Donald\AppData\Roaming\Microsoft\Signatures\Asgard.htm
Relative path:            ..\..\Signatures\Asgard.htm
Working directory:        C:\Users\Donald\AppData\Roaming\Microsoft\Signatures
NETBIOS name:             bifrost
Volume ID:                6bc0ab92-f111-496f-9067-ec c94ffa9f5
Object ID:                87349aeb-1586-11e3-be7d-24fd52566ede
MAC address:              24:fd:52:56:6e:de
```


While there are a number of paths to the target, the key one is the one labeled 'ID List' shown above. It is generated from a series of *SHITEMID* structures [9] embedded in the *SHLLINK* metadata that is used to construct the final path of the target. It is important for any SHLLINK parser to pull this out, since Windows defaults to the path described in this structure (if it exists) when resolving where the target file or directory is located.

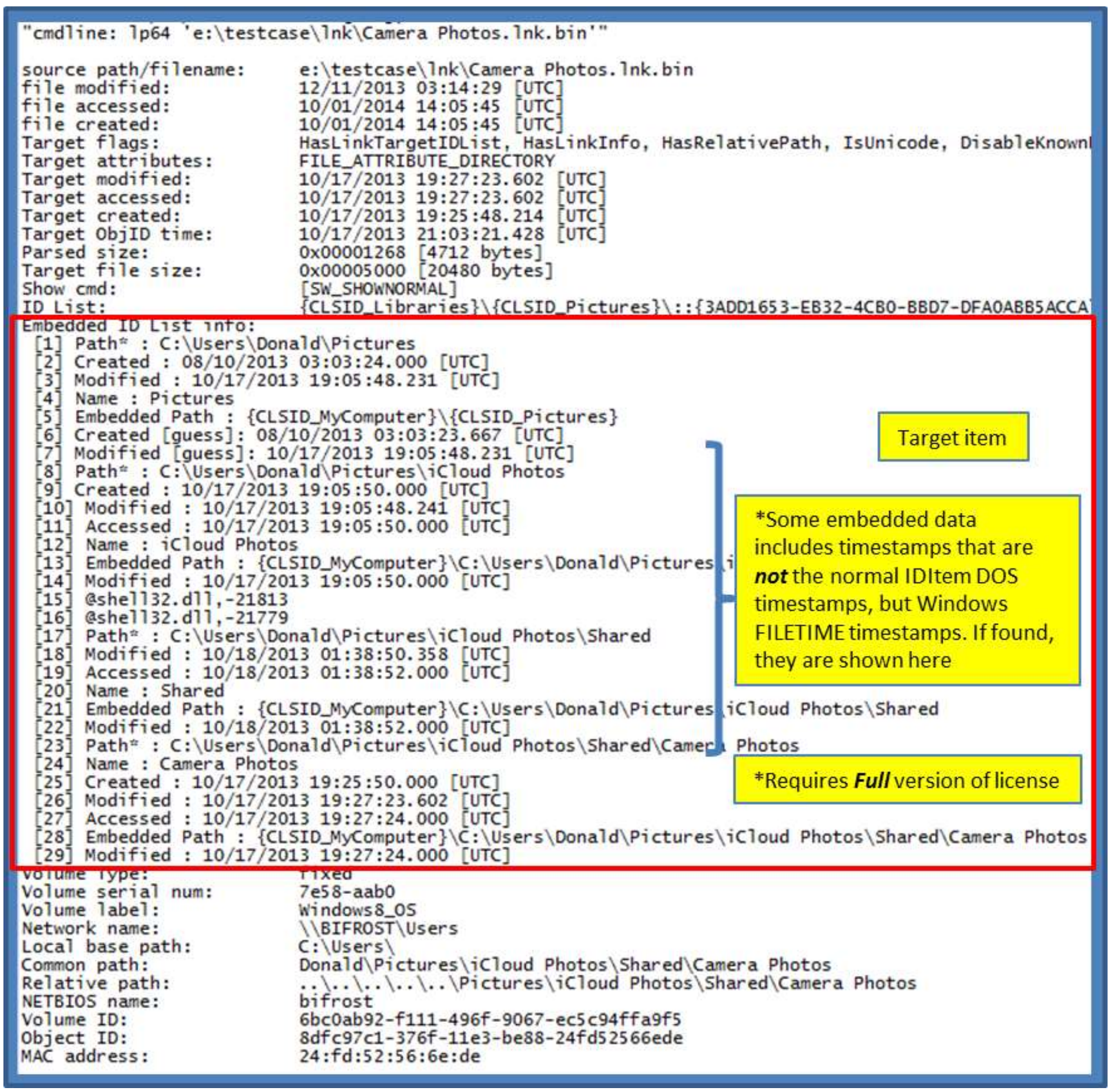
2.2 Example of breaking out more metadata from the ID List



```
"cmdline: lp64 e:\testcase\lnk\Documents.LNK.bin"

source path/filename:     e:\testcase\lnk\Documents.LNK.bin
file modified:            12/18/2013 16:13:03 [UTC]
file accessed:            10/01/2014 14:05:43 [UTC]
file created:             10/01/2014 14:05:43 [UTC]
Target flags:             HasLinkTargetIDList, IsUnicode, DisableKnownFolderAlias
Target attributes:        not specified
Target modified:          not available
Target accessed:          not available
Target created:           not available
Parsed size:              0x00000b98 [2968 bytes]
Target file size:         0x00000000 [0 bytes]
Show cmd:                 [SW_SHOWNORMAL]
ID List:                  {CLSID_MyComputer}\Donald's Windows Phone\Phone\Documents
Embedded ID List info:
 [1] \\?\usb#vid_0421&pid_0661&mi_00#6&6d096df&0&0000#                        d4f33}
 [2] {CLSID_PortableDevices}
 [3] SID-{10001,MTP Volume - 65537,31268536320}
 [4] Generic hierarchical
 [5] Serial# : MTP Volume - 65537
 [6] FuncObjId : s10001
 [7] {00010000-0514-0000-0000-000000000000}
 [8] ObjId : o1
```

If the LNK file references a file from a portable device, more detailed information can be found out about the portable device that was used. For example, if the portable device interfaced with the computer as a USB device, the LNK file may have data such as vender ID and product ID of the device that was used complete with serial number. This would be useful in tracking down that a particular device was used on that computer while accessing a file on the device. Above is an example demonstrating this type of information is available in a LNK file and was taken from the SANS 408 Donald Blake Windows 8 image.

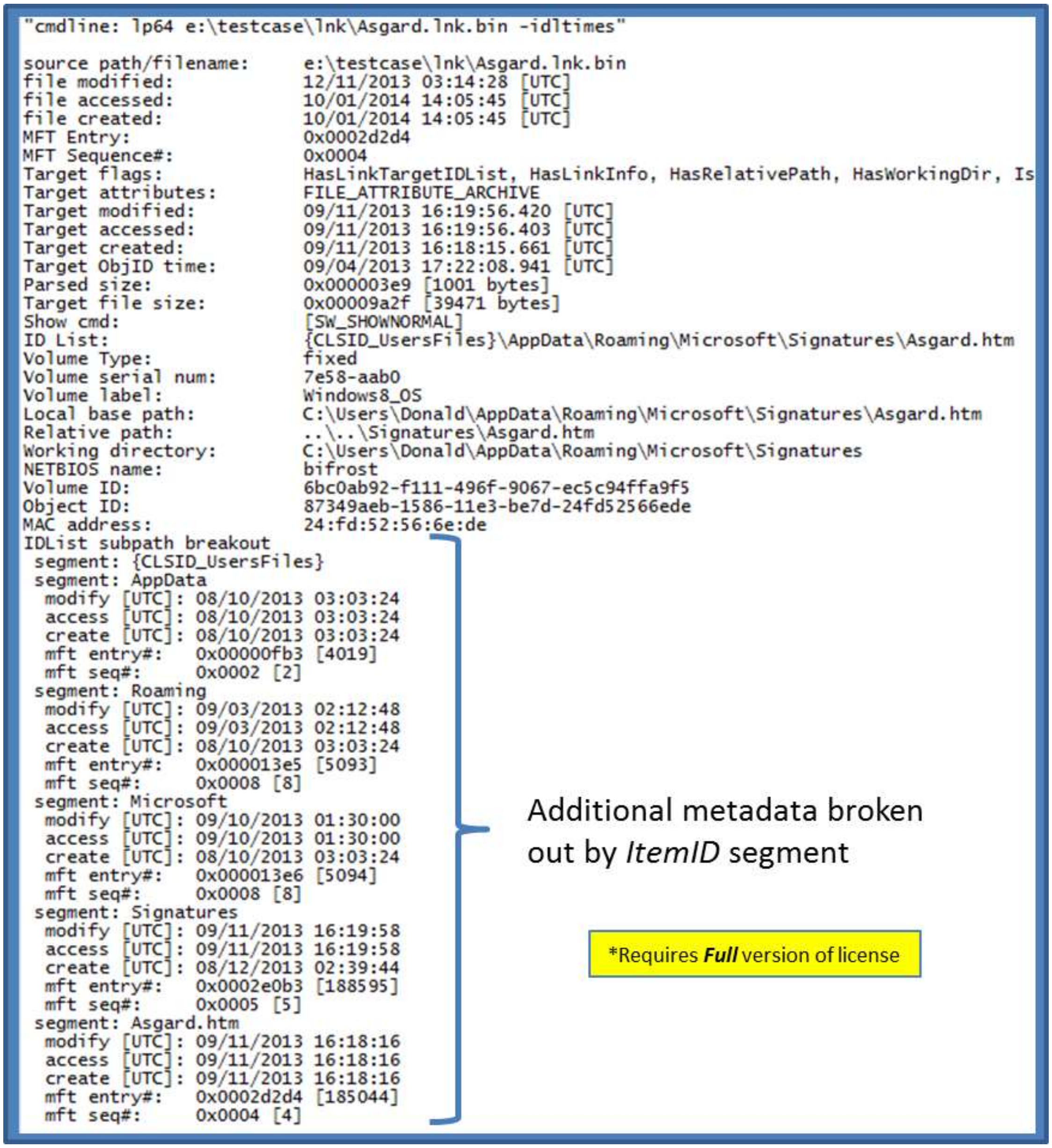
Sometimes, certain embedded information may have specific timestamps associated with the target path and item that the LNK file refers to. Normally, the ID List contains DOS timestamps, which can be extracted as well. The timestamps shown in the next example, however, represent Windows FILETIME timestamps pulled from the properties embedded into each of the nodes that make up the ID List. Below is an example of this.



```
"cmdline: lp64 'e:\testcase\lnk\Camera Photos.lnk.bin'"

source path/filename:     e:\testcase\lnk\Camera Photos.lnk.bin
file modified:            12/11/2013 03:14:29 [UTC]
file accessed:            10/01/2014 14:05:45 [UTC]
file created:             10/01/2014 14:05:45 [UTC]
Target flags:             HasLinkTargetIDList, HasLinkInfo, HasRelativePath, IsUnicode, DisableKnownI
Target attributes:        FILE_ATTRIBUTE_DIRECTORY
Target modified:          10/17/2013 19:27:23.602 [UTC]
Target accessed:          10/17/2013 19:27:23.602 [UTC]
Target created:           10/17/2013 19:25:48.214 [UTC]
Target ObjID time:        10/17/2013 21:03:21.428 [UTC]
Parsed size:              0x00001268 [4712 bytes]
Target file size:         0x00005000 [20480 bytes]
Show cmd:                 [SW_SHOWNORMAL]
ID List:                  {CLSID_Libraries}\{CLSID_Pictures}\::{3ADD1653-EB32-4CB0-BBD7-DFA0ABB5ACCA}
Embedded ID List info:
 [1] Path* : C:\Users\Donald\Pictures
 [2] Created : 08/10/2013 03:03:24.000 [UTC]
 [3] Modified : 10/17/2013 19:05:48.231 [UTC]
 [4] Name : Pictures
 [5] Embedded Path : {CLSID_MyComputer}\{CLSID_Pictures}
 [6] Created [guess]: 08/10/2013 03:03:23.667 [UTC]
 [7] Modified [guess]: 10/17/2013 19:05:48.231 [UTC]
 [8] Path* : C:\Users\Donald\Pictures\iCloud Photos
 [9] Created : 10/17/2013 19:05:50.000 [UTC]
 [10] Modified : 10/17/2013 19:05:48.241 [UTC]
 [11] Accessed : 10/17/2013 19:05:50.000 [UTC]
 [12] Name : iCloud Photos
 [13] Embedded Path : {CLSID_MyComputer}\C:\Users\Donald\Pictures\i
 [14] Modified : 10/17/2013 19:05:50.000 [UTC]
 [15] @shell32.dll,-21813
 [16] @shell32.dll,-21779
 [17] Path* : C:\Users\Donald\Pictures\iCloud Photos\Shared
 [18] Modified : 10/18/2013 01:38:50.358 [UTC]
 [19] Accessed : 10/18/2013 01:38:52.000 [UTC]
 [20] Name : Shared
 [21] Embedded Path : {CLSID_MyComputer}\C:\Users\Donald\Pictures\iCloud Photos\Shared
 [22] Modified : 10/18/2013 01:38:52.000 [UTC]
 [23] Path* : C:\Users\Donald\Pictures\iCloud Photos\Shared\Camera Photos
 [24] Name : Camera Photos
 [25] Created : 10/17/2013 19:25:50.000 [UTC]
 [26] Modified : 10/17/2013 19:27:23.602 [UTC]
 [27] Accessed : 10/17/2013 19:27:24.000 [UTC]
 [28] Embedded Path : {CLSID_MyComputer}\C:\Users\Donald\Pictures\iCloud Photos\Shared\Camera Photos
 [29] Modified : 10/17/2013 19:27:24.000 [UTC]
Volume Type:              Fixed
Volume serial num:        7e58-aab0
Volume label:             Windows8_OS
Network name:             \\BIFROST\Users
Local base path:          C:\Users\
Common path:              Donald\Pictures\iCloud Photos\Shared\Camera Photos
Relative path:            ..\..\..\..\..\Pictures\iCloud Photos\Shared\Camera Photos
NETBIOS name:             bifrost
Volume ID:                6bc0ab92-f111-496f-9067-ec5c94ffa9f5
Object ID:                8dfc97c1-376f-11e3-be88-24fd52566ede
MAC address:              24:fd:52:56:6e:de
```

2.3 Pulling out Metadata from the Segmented ItemIDs

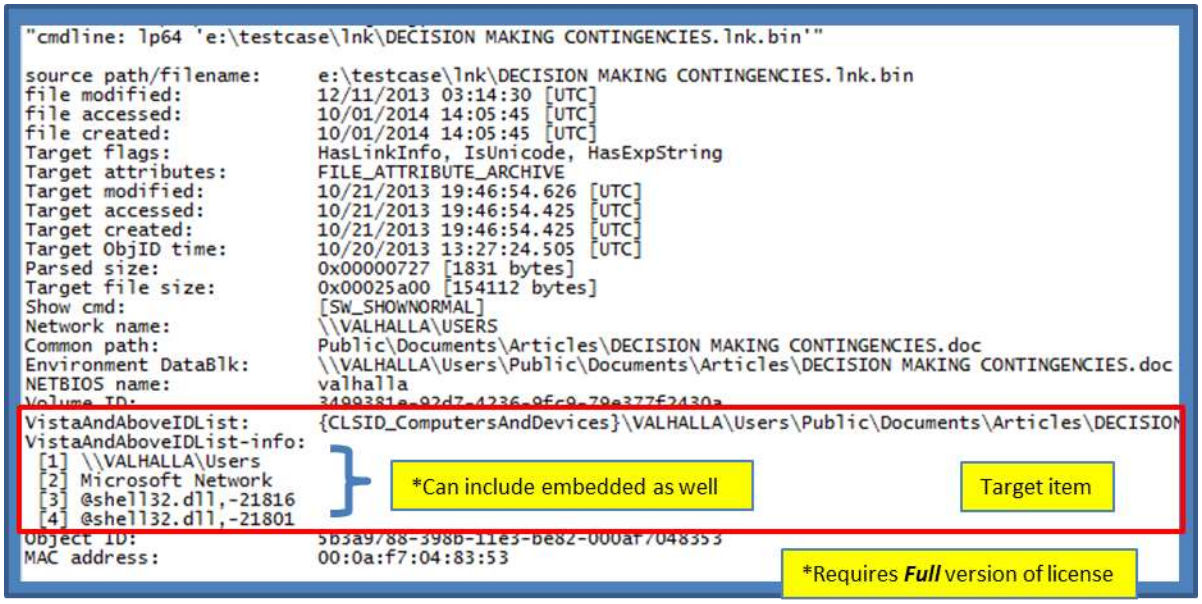
As stated earlier, Windows uses the Shell ItemID to build the path of the file specified for the link. Each ItemID can contain other information beside the segment of the path. This other information can include: (a) MAC times, MFT entry of the segment, and MFT sequence number. To pull out this additional metadata, use the ***-idltimes*** switch. Below is an example of performing this on the *Asgard.lnk* parsed earlier. The additional data outputted is highlighted below.



```
"cmdline: lp64 e:\testcase\lnk\Asgard.lnk.bin -idltimes"

source path/filename:     e:\testcase\lnk\Asgard.lnk.bin
file modified:            12/11/2013 03:14:28 [UTC]
file accessed:            10/01/2014 14:05:45 [UTC]
file created:             10/01/2014 14:05:45 [UTC]
MFT Entry:                0x0002d2d4
MFT Sequence#:            0x0004
Target flags:             HasLinkTargetIDList, HasLinkInfo, HasRelativePath, HasWorkingDir, Is
Target attributes:        FILE_ATTRIBUTE_ARCHIVE
Target modified:          09/11/2013 16:19:56.420 [UTC]
Target accessed:          09/11/2013 16:19:56.403 [UTC]
Target created:           09/11/2013 16:18:15.661 [UTC]
Target ObjID time:        09/04/2013 17:22:08.941 [UTC]
Parsed size:              0x000003e9 [1001 bytes]
Target file size:         0x00009a2f [39471 bytes]
Show cmd:                 [SW_SHOWNORMAL]
ID List:                  {CLSID_UsersFiles}\AppData\Roaming\Microsoft\Signatures\Asgard.htm
Volume Type:              fixed
Volume serial num:        7e58-aab0
Volume label:             Windows8_OS
Local base path:          C:\Users\Donald\AppData\Roaming\Microsoft\Signatures\Asgard.htm
Relative path:            ..\..\Signatures\Asgard.htm
Working directory:        C:\Users\Donald\AppData\Roaming\Microsoft\Signatures
NETBIOS name:             bifrost
Volume ID:                6bc0ab92-f111-496f-9067-ec5c94ffa9f5
Object ID:                87349aeb-1586-11e3-be7d-24fd52566ede
MAC address:              24:fd:52:56:6e:de
IDList subpath breakout
 segment: {CLSID_UsersFiles}
 segment: AppData
  modify [UTC]: 08/10/2013 03:03:24
  access [UTC]: 08/10/2013 03:03:24
  create [UTC]: 08/10/2013 03:03:24
  mft entry#:   0x00000fb3 [4019]
  mft seq#:     0x0002 [2]
 segment: Roaming
  modify [UTC]: 09/03/2013 02:12:48
  access [UTC]: 09/03/2013 02:12:48
  create [UTC]: 08/10/2013 03:03:24
  mft entry#:   0x000013e5 [5093]
  mft seq#:     0x0008 [8]
 segment: Microsoft
  modify [UTC]: 09/10/2013 01:30:00
  access [UTC]: 09/10/2013 01:30:00
  create [UTC]: 08/10/2013 03:03:24
  mft entry#:   0x000013e6 [5094]
  mft seq#:     0x0008 [8]
 segment: Signatures
  modify [UTC]: 09/11/2013 16:19:58
  access [UTC]: 09/11/2013 16:19:58
  create [UTC]: 08/12/2013 02:39:44
  mft entry#:   0x0002e0b3 [188595]
  mft seq#:     0x0005 [5]
 segment: Asgard.htm
  modify [UTC]: 09/11/2013 16:18:16
  access [UTC]: 09/11/2013 16:18:16
  create [UTC]: 09/11/2013 16:18:16
  mft entry#:   0x0002d2d4 [185044]
  mft seq#:     0x0004 [4]
```


2.4 Example of an ID List embedded into a VistaAndAboveIDList

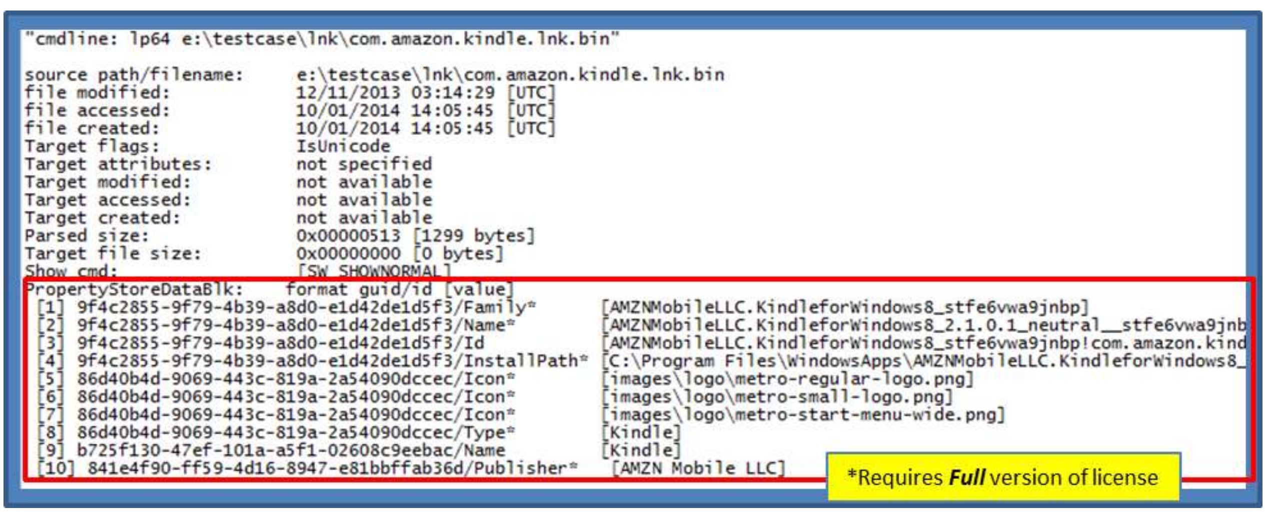
For some LNK files, the ID List is stored within the VistaAndAboveIDList data block. This, like the ID List examples previously, can have extra metadata which may provide additional insight to the target file. In some cases, the data is just redundant.



```
"cmdline: lp64 'e:\testcase\lnk\DECISION MAKING CONTINGENCIES.lnk.bin'"

source path/filename:      e:\testcase\lnk\DECISION MAKING CONTINGENCIES.lnk.bin
file modified:             12/11/2013 03:14:30 [UTC]
file accessed:             10/01/2014 14:05:45 [UTC]
file created:              10/01/2014 14:05:45 [UTC]
Target flags:              HasLinkInfo, IsUnicode, HasExpString
Target attributes:         FILE_ATTRIBUTE_ARCHIVE
Target modified:           10/21/2013 19:46:54.626 [UTC]
Target accessed:           10/21/2013 19:46:54.425 [UTC]
Target created:            10/21/2013 19:46:54.425 [UTC]
Target ObjID time:         10/20/2013 13:27:24.505 [UTC]
Parsed size:               0x00000727 [1831 bytes]
Target file size:          0x00025a00 [154112 bytes]
Show cmd:                  [SW_SHOWNORMAL]
Network name:              \\VALHALLA\USERS
Common path:               Public\Documents\Articles\DECISION MAKING CONTINGENCIES.doc
Environment DataBlk:       \\VALHALLA\Users\Public\Documents\Articles\DECISION MAKING CONTINGENCIES.doc
NETBIOS name:              valhalla
Volume ID:                 3499381e-92d7-4236-9fc9-79e377f2430a
VistaAndAboveIDList:       {CLSID_ComputersAndDevices}\VALHALLA\Users\Public\Documents\Articles\DECISION
VistaAndAboveIDList-info:
 [1] \\VALHALLA\Users
 [2] Microsoft Network
 [3] @shell32.dll,-21816
 [4] @shell32.dll,-21801
Object ID:                 5b3a9788-398b-11e3-be82-000af7048353
MAC address:               00:0a:f7:04:83:53
```


2.5 Example of a LNK file utilizing a PropertyStore Data Block

In some cases, the LNK file will make use of what is called a PropertyStore data block. This block encapsulates much metadata that could be useful in an analysis. Below is an example. For this example, this particular LNK file did not record a target file's dates or other stats common to LNK files. In this case, most of the data about the target file was started in the PropertyStore data block.



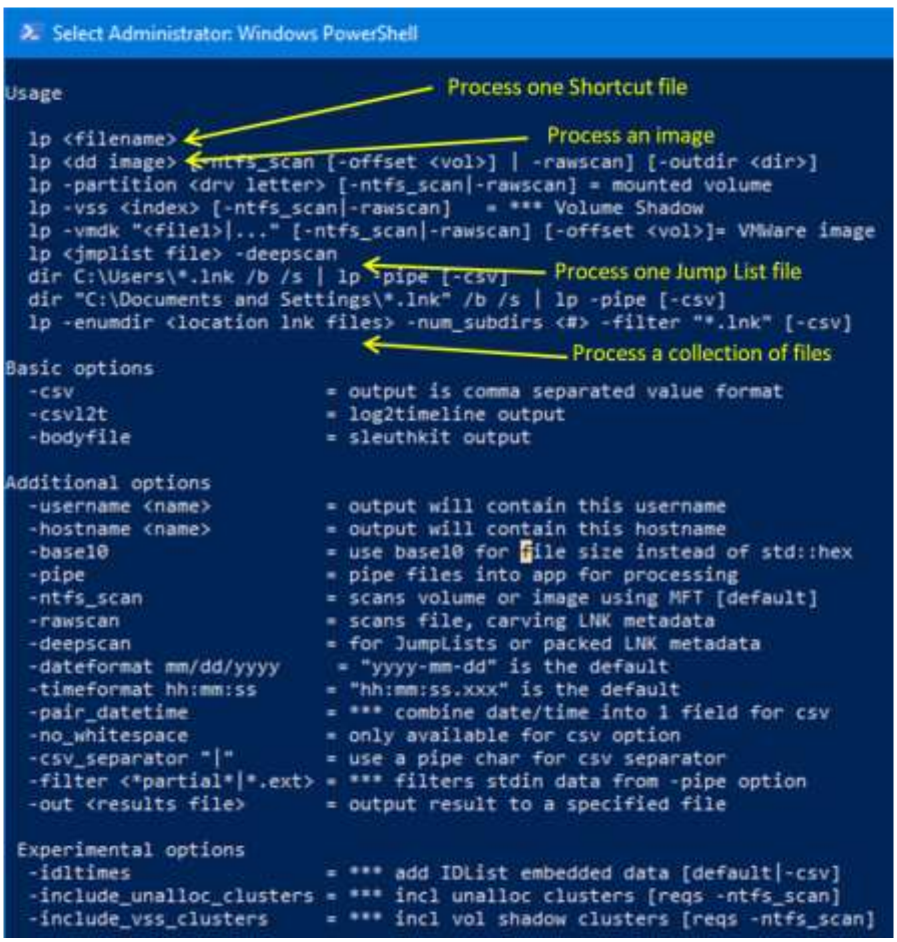
```
"cmdline: lp64 e:\testcase\lnk\com.amazon.kindle.lnk.bin"

source path/filename:      e:\testcase\lnk\com.amazon.kindle.lnk.bin
file modified:             12/11/2013 03:14:29 [UTC]
file accessed:             10/01/2014 14:05:45 [UTC]
file created:              10/01/2014 14:05:45 [UTC]
Target flags:              IsUnicode
Target attributes:         not specified
Target modified:           not available
Target accessed:           not available
Target created:            not available
Parsed size:               0x00000513 [1299 bytes]
Target file size:          0x00000000 [0 bytes]
Show cmd:                  [SW_SHOWNORMAL]
PropertyStoreDataBlk:     format guid/id [value]
 [1] 9f4c2855-9f79-4b39-a8d0-e1d42de1d5f3/Family*        [AMZNMobileLLC.KindleforWindows8_stfe6vwa9jnbp]
 [2] 9f4c2855-9f79-4b39-a8d0-e1d42de1d5f3/Name*          [AMZNMobileLLC.KindleforWindows8_2.1.0.1_neutral__stfe6vwa9jnb
 [3] 9f4c2855-9f79-4b39-a8d0-e1d42de1d5f3/Id             [AMZNMobileLLC.KindleforWindows8_stfe6vwa9jnbp!com.amazon.kind
 [4] 9f4c2855-9f79-4b39-a8d0-e1d42de1d5f3/InstallPath*  [C:\Program Files\WindowsApps\AMZNMobileLLC.KindleforWindows8_
 [5] 86d40b4d-9069-443c-819a-2a54090dccec/Icon*          [images\logo\metro-regular-logo.png]
 [6] 86d40b4d-9069-443c-819a-2a54090dccec/Icon*          [images\logo\metro-small-logo.png]
 [7] 86d40b4d-9069-443c-819a-2a54090dccec/Icon*          [images\logo\metro-start-menu-wide.png]
 [8] 86d40b4d-9069-443c-819a-2a54090dccec/Type*          [Kindle]
 [9] b725f130-47ef-101a-a5f1-02608c9eebac/Name           [Kindle]
 [10] 841e4f90-ff59-4d16-8947-e81bbffab36d/Publisher*     [AMZN Mobile LLC]
```

3 How to Use *lp*

For starters, ***lp*** is a console application. Therefore, to be able to access, and thus parse, *shortcut* files across all computer accounts, one will need to open the command prompt with administrator privileges first. Without administrator privileges, one will be restricted to only accessing your account's *shortcut* files or those common to the operating system.

One can display the menu options by typing in the executable name without parameters. A screen shot of the menu is shown below. By using the options in various ways, one can process *SHLLINK* metadata with six general '*use-cases'*: (1) processing an individual *shortcut* file, (2) carving from a captured image, (3) extracting from *Jump List* files, (4) processing a collection of files, (5) carving from a mounted volume, and (6) carving from a *VMWare* volume.

These '*use-cases'* are annotated in the screen shot below.



For output options, there are four possible formats to choose from: (1) default output, which is unstructured output. The screenshot in the previous section above is an example of what this output looks like. This information is useful if not trying to parse the artifacts into a database. (2) ***–csv*** (comma separated value) option will render the output so that all the metadata is rendered with one record per line which each field separated with a comma. The last two are: (3) ***–csvl2t*** and (4) ***-bodyfile***. Each will attempt to conform to either the *log2timeline* utility or the *SleuthKit's bodyfile* format, as appropriate.

3.1 Parsing an Individual Shortcut File

The most basic option is to parse an individual *shortcut* file. To do so, just pass the name as the parameter to ***lp***, as shown below, and the output will default to the long form shown in Section 2 above.

lp <shortcut filename>

3.2 Parsing a Captured Image for *SHLLINK* metadata

To parse an entire image of a drive that is contained in a file (eg. a 'dd' type image), one can either use the ***-rawscan*** or the ***-nfts_scan*** option. The first option ignores volume boundaries and file system internals and does a brute force scan, by looking for any *SHLLINK* signatures. For each signature found, ***lp*** will attempt to carve out any *SHLLINK* metadata. This type of scan will carve out signatures from allocated, unallocated or slack space. The second option assumes the image contains an NTFS volume, and uses the file system internals to find LNK and/or Jump List files that contain *SHLLINK* data.

lp is able to scan very large files by reading a manageable chunk from the file at a time and output the results as they are generated. So if your image is many gigabytes in size, ***lp*** should be able to process the entire image without using too much memory or system resources.

3.2.1 *rawscan* option

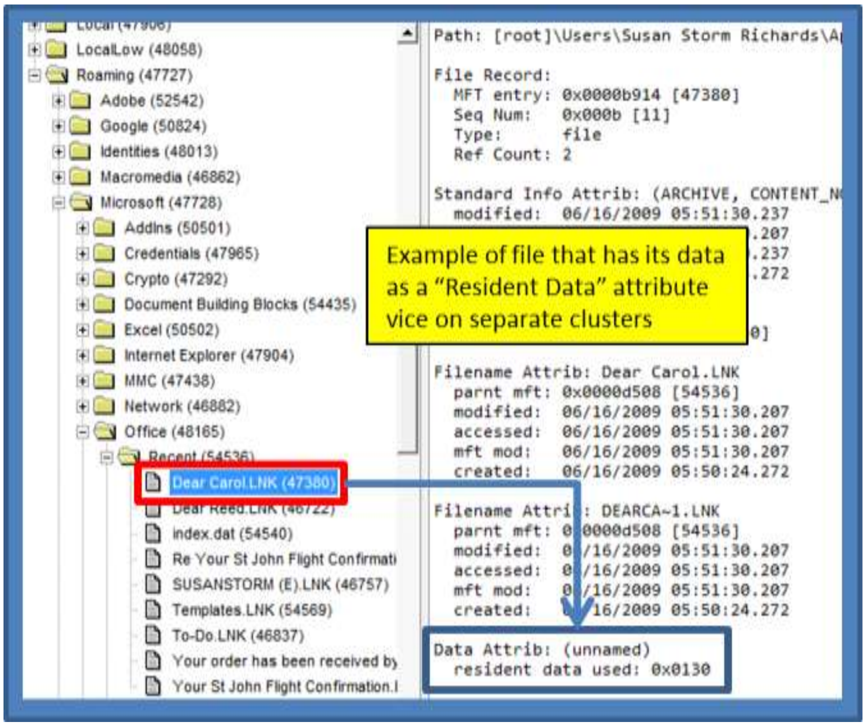
If using the ***-rawscan*** option, ***lp*** is agnostic as to the file system type, as it treats all formats the same. While this is good news in that it can work on any file system, it is also bad, in that it does not try to reconstruct files that are fragmented across non-contiguous clusters. Empirical results show, however, that since the *SHLLINK* metadata is relatively small, the fragmentation of these files is close to nil. Thus, this type of scanning/carving/parsing shows a high success rate in gathering the artifacts.

When parsing a large image, there will presumably be many *SHLLINK* entries carved out, thus it is recommended to: (a) use the ***–csv*** option to place one record per line, and (b) redirect the output into a separate file. Below is an example:

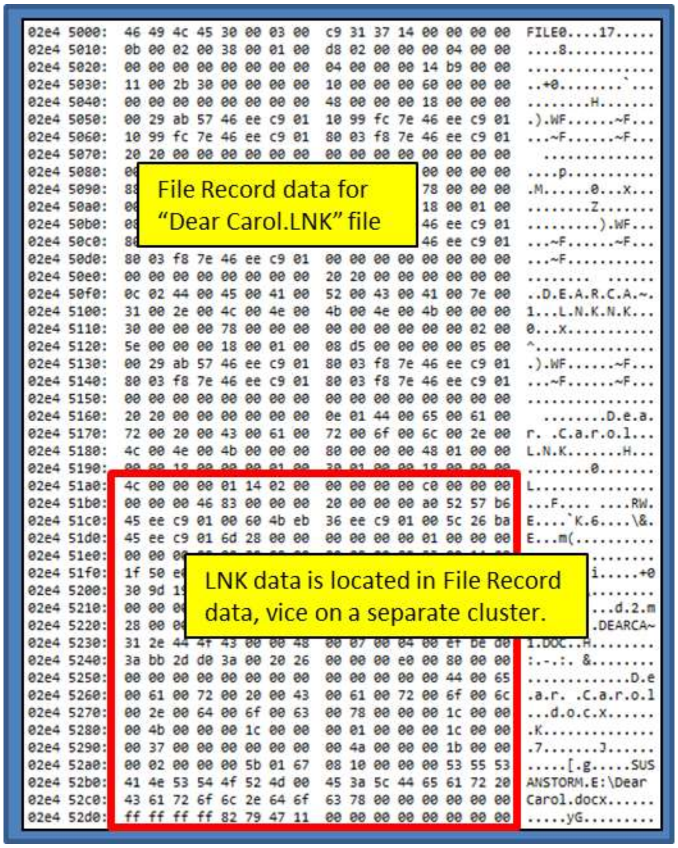
lp c:\temp\dd_imagefile.bin -rawscan -csv > results.csv

There are some caveats to be aware of when using this option, since the algorithm for locating LNK files relies on a LNK file residing on a sector boundary in the image (or volume) and tries on a best effort basis to locate LNK files embedded in certain files or file structures.

For example, there are cases in the NTFS file system where the file data may not start on a sector boundary. This happens when the NTFS *File Record* has enough space to house the data from the LNK file. This is best shown with some snapshots. For this example, we located a file that had this condition. The LNK file is called "*Dear Carol.LNK*". When looking the right side of the snapshot, the Data Attribute is highlighted to show that the data is labeled as "*Resident Data*" (vice "*Nonresident Data*"). "*Resident Data*" means the data explorer sees for the file is located within the File Record itself. Starting with version 0.62, the ***–rawscan*** option will extract these embedded LNK files from NTFS *File Records*.



If one looks at the raw *File Record* for the entry, one will see how the *File Record* attributes are packed including, in this case, the data for the file. In the snapshot below, is highlighted the actual LNK file data.



3.2.2 *ntfs_scan* option

The ***-nfts_scan*** option targets a specified mounted NTFS volume or an image with an NTFS volume. This option starts by scanning the $MFT data looking for certain files (LNK file and JumpList files) and extracts their data so it can parse the *SHLLINK* internals. This is more reliable than using the ***-rawscan*** option discussed earlier, since this option allows the data to be fully reconstructed prior to parsing it. This gives the ***lp*** tool an advantage when encountering Jump Lists, since it now can pull out the LNK chunk of data associated with the Jump List (either automatic or custom type).

If the image one want to analyze is a disk containing multiple volumes, one needs to specify the offset of the volume that will be scanned. This is done via the optional parameter ***-offset <disk offset of volume>***.

The ***-ntfs_scan*** option also allows for two sub-options to allow one to analyze the unallocated clusters associated with the volume as well as any Volume Shadow clusters. These sub-options are: ***-include_unalloc_clusters*** and ***-include_vss_clustsers***. Using these options together will yield the maximum number of *SHLLINK* data parsed.

3.3 Parsing Automatic and Custom Destinations files used for Jump Lists

Jump Lists are a new feature, starting with Windows 7. They are similar to shortcuts files in that they take one directly to the files that are used on a regular basis. They are different than the normal shortcut files in that they are more extensible in what information they display. For example, in Internet Explorer, the *Jump Lists* will display websites *frequently* visited; for Microsoft Office products like Excel, PowerPoint and Word, they will show most *recently* opened documents.

From a user's standpoint, *Jump Lists* increase one's productivity by providing quick access to the files and tasks associated with one's applications. From a forensics standpoint, *Jump Lists* are a good indicator of which files were recently opened or which websites were visited frequently.

Per Troy Larson [5], Windows derives the *Jump List* content from two sets of *Destination* files:

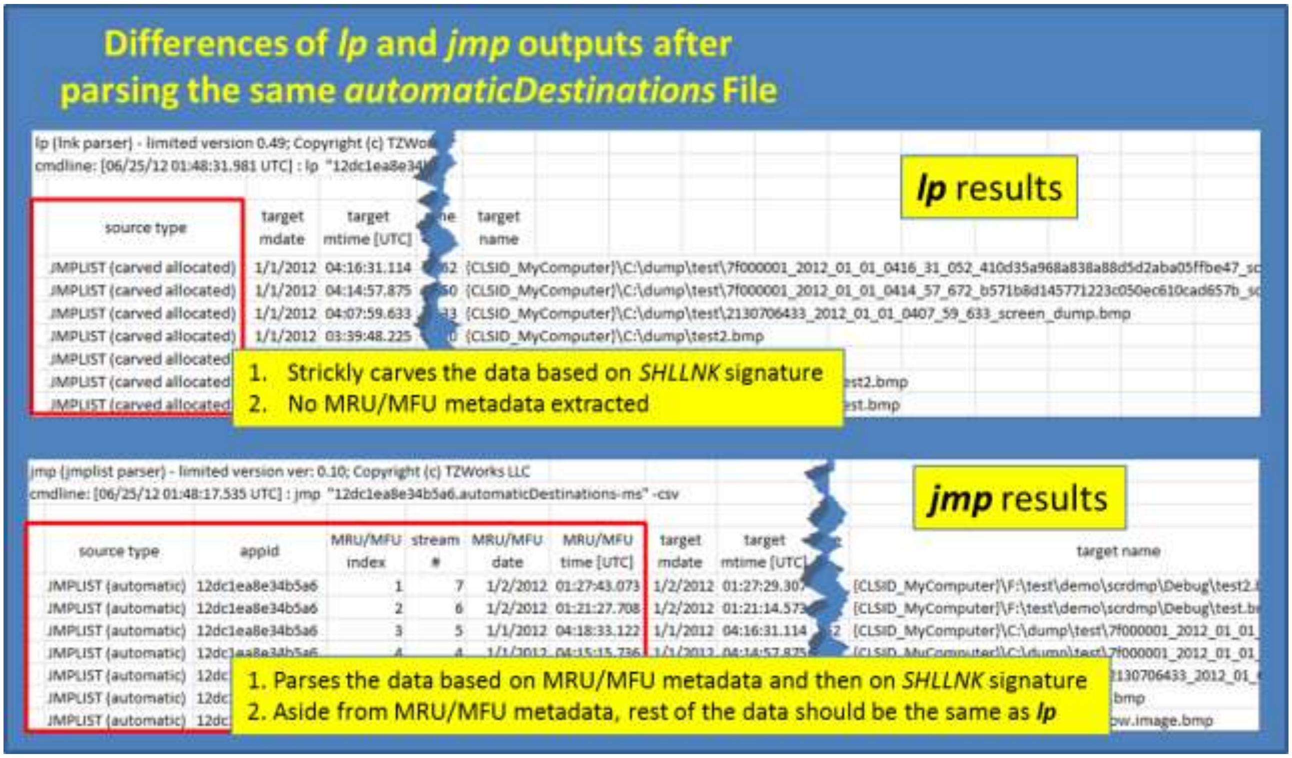
a. %APPDATA%\Microsoft\Windows\Recent\AutomaticDestinations\[*AppID*].automaticDestinations-ms
b. %APPDATA%\Microsoft\Windows\Recent\CustomDestinations\[*AppID*].customDestinations-ms

%APPDATA% is resolved to *C:\Users\<user account>\AppData\Roaming.* One can see that each *user account* (or profile) has its own set of *Destination* files.

For most *automatic Destinations* type files, ***lp*** can find and parse the *SHLLINK* metadata with no special command line options (eg. using just the default settings). This is because the *automatic Destinations* type files have a compound file signature, which is built into the ***lp*** scanning engine. ***lp*** will recognize this signature, reconstruct the allocated/unallocated sectors within the compound file and scan the chunks appropriately. On the other hand, the *custom Destination* type files only have *SHLLINK* signatures which do not necessarily occur on sector boundaries. Therefore, to assist ***lp***, to parse this type of file, one invokes the ***–deepscan*** switch. This tells ***lp*** to scan in a mode that is in-between a normal *LNK* file scan and a captured image type scan. This switch has no effect on normal *shortcut* files, so it can be used to handle both *shortcut* files as well as *automatic/custom Destinations* files.

While ***lp*** does a good job at pulling out the *SHLLINK* metadata from both *automatic* and *custom Destination* type files, it ***does not*** attempt to parse the MRU/MFU data from the a*utomatic Destinations* files. To parse these files in a complete fashion, one can use the ***jmp*** tool from *TZWorks*. The ***jmp*** tool understands how to parse the both types of *Destinations* files in a manner to extract all pertinent metadata for the investigator.

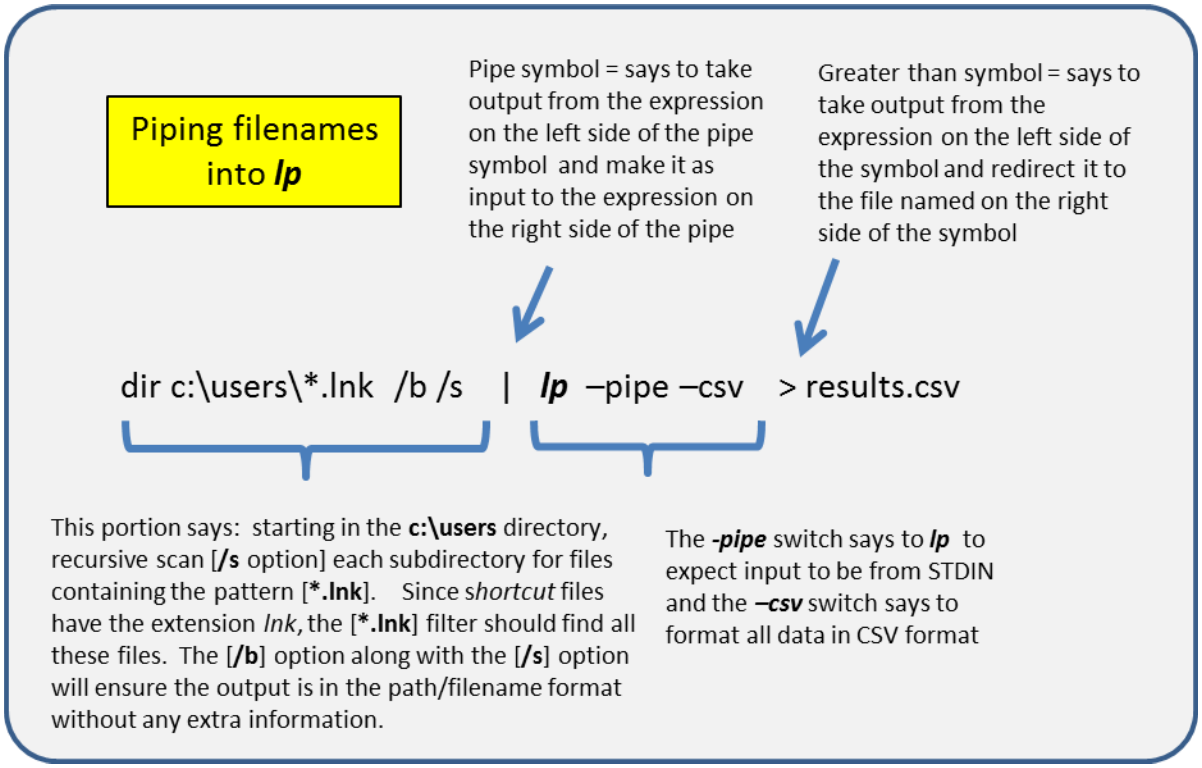
Below is a comparison of the outputs of running the ***lp*** tool and the ***jmp*** tool against the same *automatic Destinations* file. This output is representative of the differences between the two tools. For more information about the ***jmp*** tool, one can download and review the readme file for the tool.



3.4 Parsing a Collection of Files

Sometimes one just wants to parse a bunch of *shortcut* files that are in a directory or a collection of subdirectories. Compared to the partition scan discussed above, this option is much faster. The disadvantage with this approach over the partition scan, is that one does not get artifacts that have been deleted and are still in unallocated or slack space.

To use this option, one will make use of the operating systems ability to pipe data from one application's output to another application's input. In this case, the source of the data will be the Windows shell command ***dir***. By adding some special options to the ***dir*** command, one can output only the path/filename without any extra data. This result will be consumed by ***lp,*** and each path/filename passed in will be analyzed. To invoke this behavior in ***lp***, one will use the ***–pipe*** switch. The annotated figure below explains how the syntax of the command is composed.



If one cannot use the ***-pipe*** option, one can use the experimental ***-enumdir*** option, which has similar functionality with more control. The ***-enumdir*** option takes as its parameter the folder to start with. It also allows one to specify the number of subdirectories to evaluate using the ***-num_subdirs <#>*** sub-option.

3.5 Parsing an Active Volume [Experimental Option]

A variant of parsing a captured image is to parse an *active* Windows partition or a mounted volume on Linux. The Windows version is invoked by using the ***-partition <drive letter>*** option. On Linux, this is handled by passing in the device name of the disk and/or volume as the filename without the use of the ***–partition*** keyword. The other option that needs to be set is to identify whether to use the ***–rawscan*** or ***-ntfs_scan*** option.

Below is an example of ***lp*** carving out *SHLLINK* signatures from a USB drive mounted as drive H for Windows and */dev/sbd1* on Linux.

lp -partition H [-nfts_scan | -rawscan] -csv > results.csv [Windows version]

lp /dev/sdb1 –csv [-nfts_scan | -rawscan] > results.csv [Linux version]

To find where the drive is mounted on Linux or Mac, one can use the built-in tool ***df*** to enumerate what devices are used for the mount. While the ***df*** command is to display free disk space, it does this by displaying all the devices mounted followed by their statistics. For the Mac OS-X case, one could also use the ***diskutil list*** to enumerate all drives and volumes mounted.

The Mac OS-X has an additional nuance in that one needs to specify *raw* I/O vice the standard *buffered* I/O. So for the example above, if */dev/sdb1* was specified as the device for the drive, then one would issue */dev/**r**sdb1*.

3.6 Parsing a *VMWare* volume

Occasionally, it is useful to analyze a *VMWare* image, both from a forensics standpoint, as well as, from a testing standpoint. When analyzing different operating systems, and different configurations, a virtual machine is extremely useful in testing out different boundary conditions. This option is still considered *experimental* since it has only been tested on a handful of VMDK 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 [8].

When working with virtual machines, the capability to handle snapshot images is important. When processing a *VMWare* snapshot, one needs to include the parent snapshot/image as well as any descendants.

lp 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>" –offset <volume offset>***). To aid the user in figuring out exactly the chain of descendant images, ***lp*** can take any VMDK file (presumably the VMDK of the snapshot one wishes to analyze) and determine what the descendant chain is. Finally, ***lp*** will suggest a chain to use. In the syntax above there is also the ***-offset*** parameter. Without specifying the volume offset, the ***-vmdk*** option will try to find the first *NTFS* volume and analyze that one. Therefore, if your *VMDK* volume has multiple *NTFS* volumes and you wish to look at something other than the first one, you would need to explicitly tell ***lp*** to do that by specifying the ***-offset*** parameter.

3.7 Parsing Volume Shadows

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

To tell ***lp*** to look at a Volume Shadow, one needs to use the ***-vss <index of volume shadow>*** option. This points ***lp*** at the appropriate Volume Shadow and it starts analyzing the various user directories for LNK files, and if any are found, parses them. Below is an example of traversing Volume Shadow Copy #1 and rendering the CSV results to a file called vss1_out.csv.

lp -vss 1 [-ntfs_scan | -rawscan] -csv > vss1_out.csv

If one only wants to look for a particular LNK file in a particular Volume Shadow, one can use the keyword %vss% . Below is an example of telling lp to parse the LNK file at Volume Shadow Copy #1.

lp %vss%1\users\testuser\AppData\Roaming\Microsoft\Windows\Recent\out.txt.lnk

The %vss% keyword, in combination with the number that follows the keyword, is expanded internally to point to the proper Volume Shadow.

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:

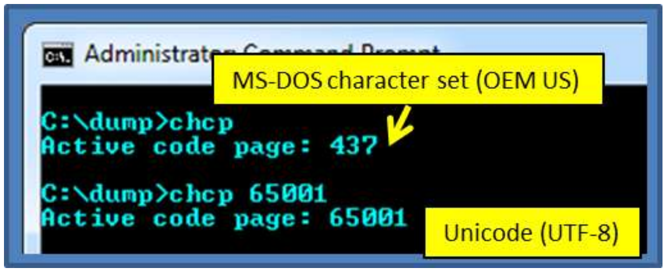
Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1
Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy2
...

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

3.8 Parsing non-ASCII character sets

Since ***lp*** was built to use UTF-8 internally, it can handle non-ASCII character sets without any modifications. However, when dealing with LNK files, there are instances that reading in a non-ASCII character filename can be problematic from the command prompt. This section discussion some of the non-ASCII character options available to Windows users running ***lp***.

When using the ***–pipe*** option with Windows, one can tell the data that is inputted into an application (such as ***lp***) to be *UTF-8* by changing the active code page from the default one to *UTF-8*. This can be done via the command, ***chcp 65001***, and then one can pipe in a directory of files into ***lp*** and standard input will yield the path/filename to display Unicode (UTF-8) format. Below is a screen shot of using the ***chcp*** command in Windows.



4 Known Issues

- ***lp*** doesn't parse some of the *SHLLINK* structures documented in the Microsoft specification. As time permits, future versions will incorporate incremental capabilities to handle these structures.
- 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 commas in its name are changed to semicolons. For the default (non-csv) output, no changes are made to the data.
- For carving options from an image or a volume, ***lp*** can encounter boundary conditions that we did not experience during our testing phase. For these cases, ***lp*** will most likely crash. As we discover new untested boundary conditions new updates will be rolled out.
- [Note: this issue does not apply to ***-ntfs_scan*** option] For Linux and Mac builds, the *file cdate & time* reported in the output is the date and time of the metadata change of the file (not the creation time of the file). This behavior is different in Windows, where the *file cdate & time* reported in the output is the date and time of the creation of the file.
- (Windows only) When processing filenames with characters that are not ASCII, one option is to change the code page of the command window from the default code page to UTF-8. This can be done at the command window via the command: ***chcp 65001***

5 Available Options

Option	Description
-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

<table>
<tr><td></td><td>needs to set the environment variable TZ=UTC.</td></tr>
<tr><td>-base10</td><td>Ensure all size/address output is displayed in base-10 format vice hexadecimal format. Default is hexadecimal format</td></tr>
<tr><td>-username</td><td>Option is used to populate the output records with a specified username. The syntax is -username <name to use>.</td></tr>
<tr><td>-hostname</td><td>Option is used to populate the output records with a specified hostname. The syntax is -hostname <name to use>.</td></tr>
<tr><td>-pipe</td><td>Used to pipe files into the tool via STDIN (standard input). Each file passed in is parsed in sequence.</td></tr>
<tr><td>-enumdir</td><td>Experimental. Used to process files within a folder and/or subfolders. Each file is parsed in sequence. The syntax is -enumdir <folder> -num_subdirs <#>.</td></tr>
<tr><td>-filter</td><td>Filters data passed in via STDIN via the -pipe option. The syntax is -filter <"*.ext | *partialname* | ...">. The wildcard character '*' is restricted to either before the name or after the name.</td></tr>
<tr><td>-rawscan</td><td>Scan a large file or captured image looking for SHLLINK signatures and when found parse them. This option is not meant to be used for individual shortcut files</td></tr>
<tr><td>-outdir</td><td>Used in conjunction with the -rawscan option to store carved LNK files to the specified directory. The syntax is -rawscan -outdir <directory>.</td></tr>
<tr><td>-deepscan</td><td>Added just for Destinations files used for Jump Lists. Since Destinations can have many SHLLINK signatures embedded into one file, this option handles parsing these types of files correctly. Note: the -deepscan option and the -rawscan option cannot be used together.</td></tr>
<tr><td>-ntfs_scan</td><td>Scan a NTFS volume looking for SHLLINK signatures and when found parse them. The basic option scans the $MFT data. There are sub-options to do a more extensive scan: (a) -include_unalloc_clusters and
(b) -include_vss_clusters. The first will scan the unallocated clusters and carve out any LNK data and the second will scan the Volume shadow clusters and carve out any LNK data.</td></tr>
<tr><td>-partition</td><td>Used to scan a mounted Windows volume for SHLLINK signatures and parse them. When this option in invoked, the option -rawscan is implicitly invoked.</td></tr>
</table>

	Since this option is traversing a mounted volume at the cluster level, it requires the tool to be running at administrative privileges. The syntax is ***-partition <drive letter>***.
-vmdk	Extract artifacts from a VMWare monolithic NTFS formatted volume. The syntax is ***-vmdk "disk"***. When this option in invoked, the **-rawscan** option is implicitly invoked. For a collection of VMWare disks that include snapshots, one can use the following syntax: ***-vmdk "disk1 | disk2 | ..."***
-vss	Experimental. Parse LNK 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.
-idltimes	Experimental. Shell item identifiers are grouped together to form a path. Each Item ID can have embedded in it an associated MAC timestamps as well as MFT entry number for the segment of the path that creates the final path. Using this option will display any additional metadata associated with each segment (or Item ID) in the list
-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, 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.xxxxxxxxx"***, 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
-out	Output the data to the specified file. The syntax is ***-out <results file>***.
-utf8_bom	All output is in Unicode UTF-8 format. If desired, one can prefix an UTF-8 *byte order mark* to the 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. SANS Institute. Forensics 408 course (taken in Jan 2010)
2. [MS-SHLLINK]: Shell Link (.LNK) Binary File Format, 11/12/2010, sourced from Microsoft Corporation. http://msdn.microsoft.com/en-us/library/dd871305(v=prot.13).aspx
3. Jesse Hager "The Windows Shortcut File Format", Available at http://www.i2slab.com/Papers/The_Windows_Shortcut_File_Format.pdf.

4. http://www.forensicswiki.org/wiki/LNK
5. Windows 7 Jump Lists, windows7forensics-jumplists-rv3-public-110606164708-phpapp01.pptx, Troy Larson PowerPoint charts.
6. *SleuthKit* Body-file format, http://wki.sleuthkit.org
7. *Log2timeline* CSV format, http://log2timeline.net/
8. *VMWare* Virtual Disk Format 1.1 Technical Note, www.vmware.com
9. SHITEMID structure. http://msdn.microsoft.com/en-us/library/windows/desktop/bb759800(v=vs.85).aspx
10. RFC 4122, A Universally Unique Identifier (UUID) URN Namespace, published July 2005.