TZWorks® ShellBag Parser (*sbag*) Users Guide

Abstract

sbag is a standalone, command-line tool used to extract Shellbag artifacts from Windows user account registry hives. It can operate on a live target hives or on separately captured registry hives. All artifacts can be outputted in one of three formats for easy inclusion with other forensics artifacts.

www.tzworks.com

Contact Info: info@tzworks.com

*Document applies to v0.80 of **sbag***

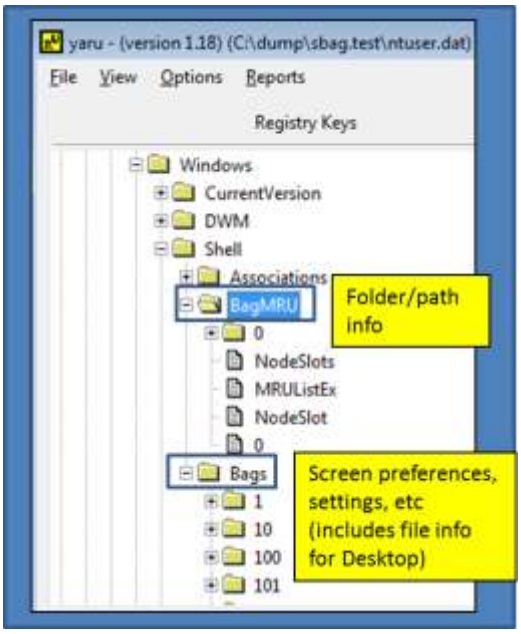
Updated: Apr 25, 2025

Table of Contents

TZWorks® ShellBag Parser (*sbag*) Users Guide

Webpage: http://www.tzworks.com/prototype_page.php?proto_id=14
Contact Information: info@tzworks.com

1 Introduction



sbag is a command line version of a Windows registry parser that targets the Shellbag subkeys to pull useful directory and file artifacts to help identify user activity. There are binaries available for Windows, Linux and Mac OS-X. The Windows version allows one to parse hives resident from a live system.

As background, the ShellBag information is a set of subkeys in a user registry hive (eg. *ntuser.dat* and *usrclass.dat* files) used by the Windows operating system to track user window viewing preferences. It does this by storing various Windows Explorer settings that relates to dimensions, settings, etc. This allows one to reopen the same folder at a later time with the settings from the previous time. Each user will have separate preferences for folders, and therefore, these settings are stored in the appropriate user hive.

Since the *ShellBag* subkeys store various metadata on how Windows Explorer items were arranged, and since they are recorded for each user, from a computer forensics standpoint, one can parse the data and pull out various pieces of information that relate to user interaction. When combined with other available computer artifacts, it could provide a more complete picture of what files were accessed, or deleted by the user, and from what storage device they were accessing (could be either an internal, external or network storage device). The *'ShellNoRoam\BagXxx'* key(s) has data for local folders and the *'Shell\BagXxx'* key(s) has data for the remote folders.

The registry subkeys that ***sbag*** evaluates include the following:

- NTUSER.DAT\Software\Microsoft\Windows\Shell\BagMRU
- NTUSER.DAT\Software\Microsoft\Windows\Shell\Bags
- NTUSER.DAT\Software\Microsoft\Windows\ShellNoRoam\BagMRU
- NTUSER.DAT\Software\Microsoft\Windows\ShellNoRoam\Bags

- UsrClass.DAT\Local Settings\Software\Microsoft\Windows\Shell\BagMRU
- UsrClass.DAT\Local Settings\Software\Microsoft\Windows\Shell\Bags
- UsrClass.DAT\Local Settings\Software\Microsoft\Windows\ShellNoRoam\BagMRU
- UsrClass.DAT\Local Settings\Software\Microsoft\Windows\ShellNoRoam\Bags

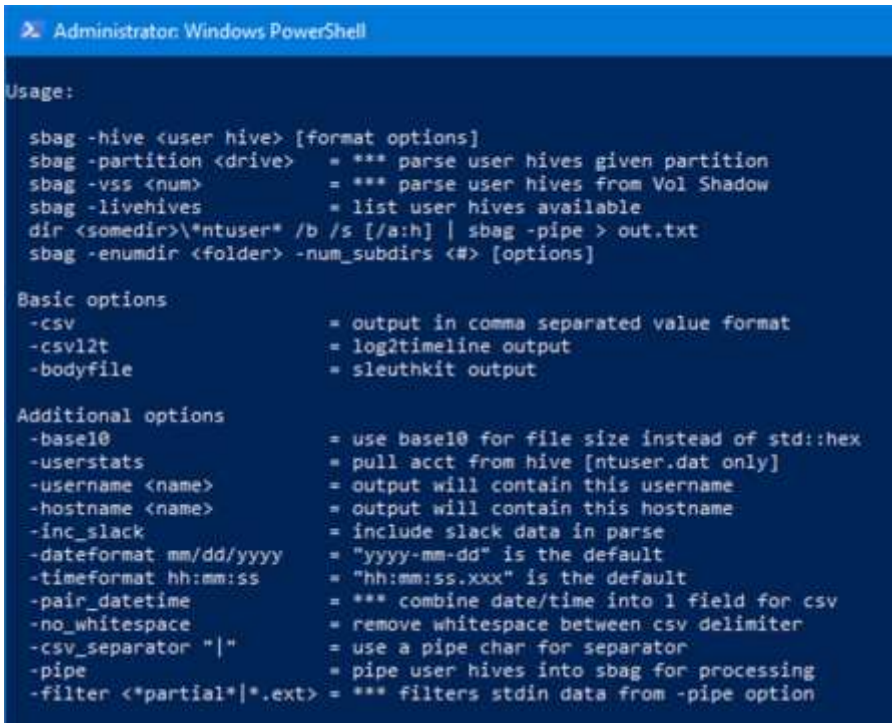
On Vista and Windows 7, the *UsrClass.dat* hive is new from the older Windows XP and is located in the *C:\Users\<user>\AppData\Local\Microsoft\Windows* directory.

2 How to Use *sbag*

sbag is a console application, and thus, to use this tool on a live system, one will need to open the command prompt with administrator privileges first. One can display the menu options by typing in the executable name without parameters. A screen shot of the menu is shown below.

While the menu shows a number of different options, the only required parameter to pass in is the user hive one wishes to extract *shellbag* artifacts from. The available options include: (a) annotating username and/or hostname to the output (version 0.23+), (b) specifying what type of format one wishes to put the output data in, (c) whether to extract data from cell slack space (version 0.24+), (d) miscellaneous and date/time format options (version 0.29+), (e) the ability to pipe in hives for analysis (version 0.33+), and (f) the ability to parse all the user hives by either pointing to a drive letter or a Volume Shadow copy (version 0.38).

The output options include: (a) the default output, where each record is on a separate line and each field is separated by the pipe character, (b) the SleuthKit body-file format [5] and (c) the log2timeline CSV (comma separated value) format. [6]



2.1 Processing Individual Hives

Below is an example of parsing a user hive in an off-line manner. For this example, it assumes a user hive was extracted to the c:\dump directory beforehand. To parse the hive, one could then invoke the following command:

sbag -hive c:\dump\ntuser.dat > results.txt

Since the output that is generated is very wide, it is recommended that one redirect the output of the command into a results file as shown above. Then, it can be reviewed in any text editor by turning off the word wrap option to see each record on a separate line. If one wants to take advantage of the Comma Separate Value (CSV) format that is easily opened with any spreadsheet application, one could use the ***-csv*** option, like below:

sbag -hive c:\dump\ntuser.dat -csv > results.csv

The only difference in the results between the normal output and the CSV output is the CSV processed output is checked for any occurrences of commas. If any commas are found, they are changed into semicolons so as to not disrupt the CSV format for separating the fields. Thus, if one needs exact data without any possible modifications, one should choose the normal (default) output. The normal (default) output uses the pipe character '|' for a delimiter which does not conflict with any valid Windows filename syntax.

2.2 Processing Volume Shadow Copies (Individual Hive or Entire Volume)

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 make it easier with the syntax, we've built in some shortcut syntax to access a specified Volume Shadow copy, via the ***%vss%*** keyword. This internally gets expanded into *\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy*. Thus to access index 1 of the volume shadow copy, one would prepend the keyword and index, like so, ***%vss%1*** to the normal path of the hive. For example, to access a user hive located in the *testuser* account from the *HarddiskVolumeShadowCopy1*, the following syntax can be used:

sbag -hive %vss%1\Users\testuser\ntuser.dat > results.txt

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, the keywords one needs to look for are names that look like this:

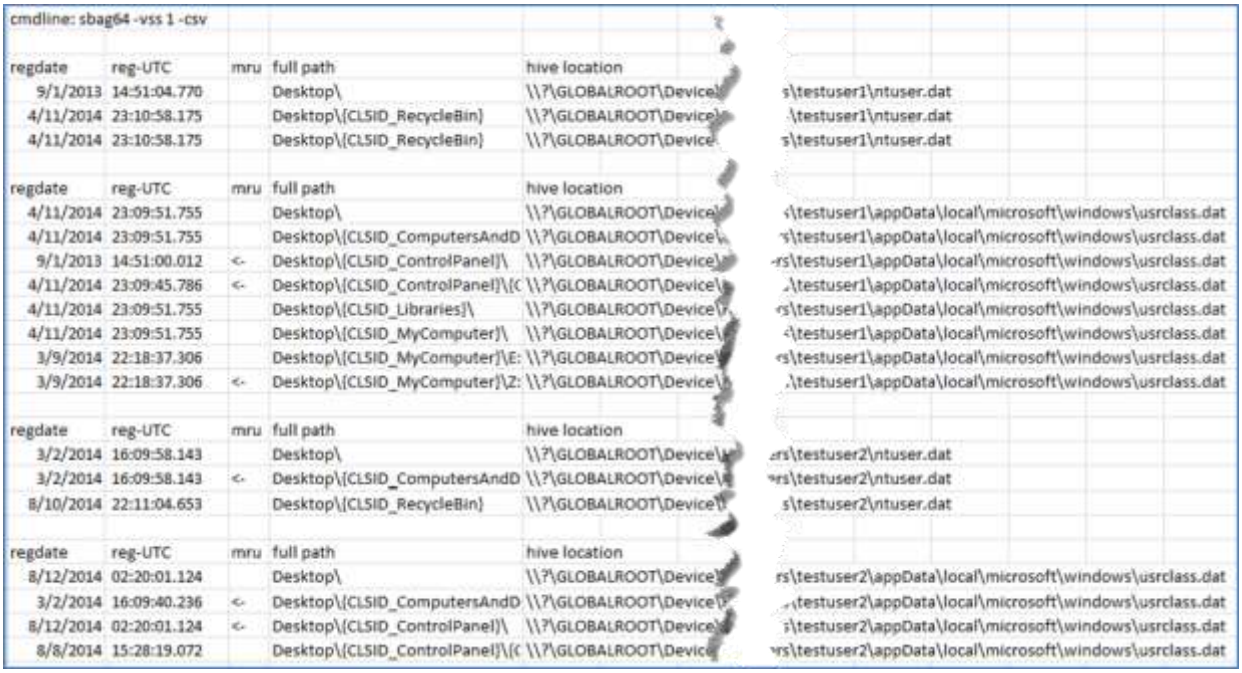
Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1
Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy2
. . .

From the above, notice the number after the word *HarddiskvolumeShadowCopy*. It is this number that is appended to the %vss% keyword.

In addition, there is the ability to automatically find and parse *all* the *ntuser.dat* and *usrclass.dat* hive on a given Volume Shadow, by using the ***-vss <index of Volume Shadow>*** option. To invoke this, just pass in which Volume Shadow copy you wish to analyze. One does not need to specify a hive since it will find and parse all the hives in the user directories. For example, to analyze copy 1:

sbag -vss 1 -csv > results.txt

When using the default output option or the normal CSV option, the output will include a separate header for each hive found and parsed. The hive location will be annotated with the volume shadow copy symbolic name and hive path (see below). If using the ***-csvl2t*** option, all the records will be integrated under one header.



cmdline: sbag64 -vss 1 -csv

regdate	reg-UTC	mru	full path	hive location
9/1/2013	14:51:04.770		Desktop\	\\?\GLOBALROOT\Device\ s\testuser1\ntuser.dat
4/11/2014	23:10:58.175		Desktop\{CLSID_RecycleBin}	\\?\GLOBALROOT\Device\ \testuser1\ntuser.dat
4/11/2014	23:10:58.175		Desktop\{CLSID_RecycleBin}	\\?\GLOBALROOT\Device s\testuser1\ntuser.dat

regdate	reg-UTC	mru	full path	hive location
4/11/2014	23:09:51.755		Desktop\	\\?\GLOBALROOT\Device\ \testuser1\appData\local\microsoft\windows\usrclass.dat
4/11/2014	23:09:51.755		Desktop\{CLSID_ComputersAndD	\\?\GLOBALROOT\Device\ s\testuser1\appData\local\microsoft\windows\usrclass.dat
9/1/2013	14:51:00.012	<-	Desktop\{CLSID_ControlPanel}\	\\?\GLOBALROOT\Device\ rs\testuser1\appData\local\microsoft\windows\usrclass.dat
4/11/2014	23:09:45.786	<-	Desktop\{CLSID_ControlPanel}\{C	\\?\GLOBALROOT\Device\ .\testuser1\appData\local\microsoft\windows\usrclass.dat
4/11/2014	23:09:51.755		Desktop\{CLSID_Libraries}\	\\?\GLOBALROOT\Device\ rs\testuser1\appData\local\microsoft\windows\usrclass.dat
4/11/2014	23:09:51.755		Desktop\{CLSID_MyComputer}\	\\?\GLOBALROOT\Device\ \testuser1\appData\local\microsoft\windows\usrclass.dat
3/9/2014	22:18:37.306		Desktop\{CLSID_MyComputer}\E:	\\?\GLOBALROOT\Device\ rs\testuser1\appData\local\microsoft\windows\usrclass.dat
3/9/2014	22:18:37.306	<-	Desktop\{CLSID_MyComputer}\Z:	\\?\GLOBALROOT\Device\ .\testuser1\appData\local\microsoft\windows\usrclass.dat

regdate	reg-UTC	mru	full path	hive location
3/2/2014	16:09:58.143		Desktop\	\\?\GLOBALROOT\Device\ rs\testuser2\ntuser.dat
3/2/2014	16:09:58.143	<-	Desktop\{CLSID_ComputersAndD	\\?\GLOBALROOT\Device\ rs\testuser2\ntuser.dat
8/10/2014	22:11:04.653		Desktop\{CLSID_RecycleBin}	\\?\GLOBALROOT\Device\ s\testuser2\ntuser.dat

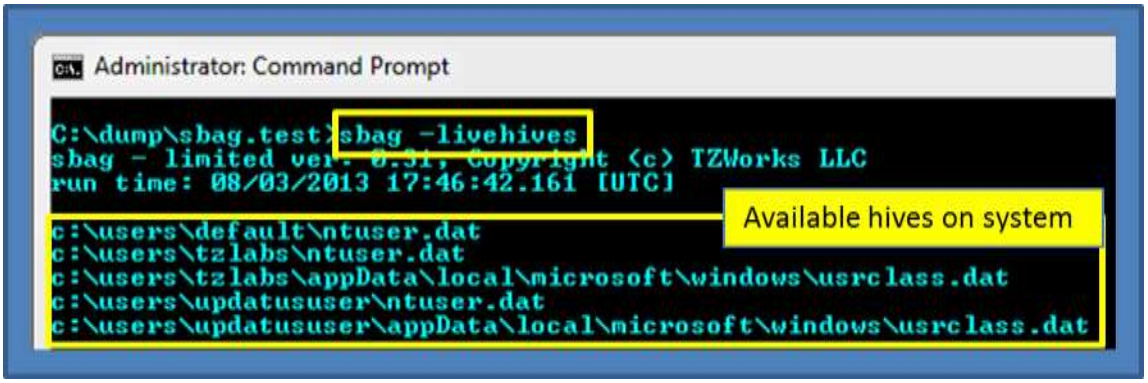
regdate	reg-UTC	mru	full path	hive location
8/12/2014	02:20:01.124		Desktop\	\\?\GLOBALROOT\Device\ rs\testuser2\appData\local\microsoft\windows\usrclass.dat
3/2/2014	16:09:40.236	<-	Desktop\{CLSID_ComputersAndD	\\?\GLOBALROOT\Device\ testuser2\appData\local\microsoft\windows\usrclass.dat
8/12/2014	02:20:01.124	<-	Desktop\{CLSID_ControlPanel}\	\\?\GLOBALROOT\Device\ s\testuser2\appData\local\microsoft\windows\usrclass.dat
8/8/2014	15:28:19.072		Desktop\{CLSID_ControlPanel}\{C	\\?\GLOBALROOT\Device rs\testuser2\appData\local\microsoft\windows\usrclass.dat

2.3 Processing all User Hives on Partition

Added with version 0.38, one can process all the user hives on a specified volume (including the live system volume). This option is invoked with the following syntax: ***-partition <drive letter>***. This option only looks in the normal *users'* directories to find which hives are available, and then proceeds to process those hives. It is useful if mounting a collected image of a system volume as another drive letter.

2.4 Enumerating Available User Hives

To just enumerate which hives are available for processing *ShellBag* artifacts from a live system, one can use the ***-livehives*** option. Once the path of the location of the user hives are known, one can parse any of the desired hives by passing in the path of the active hive. For this type of live processing, ***sbag*** will take care of taking a snapshot of the requested hive by reading the appropriate raw NTFS clusters and then proceed to extract the artifacts from the snapshot.

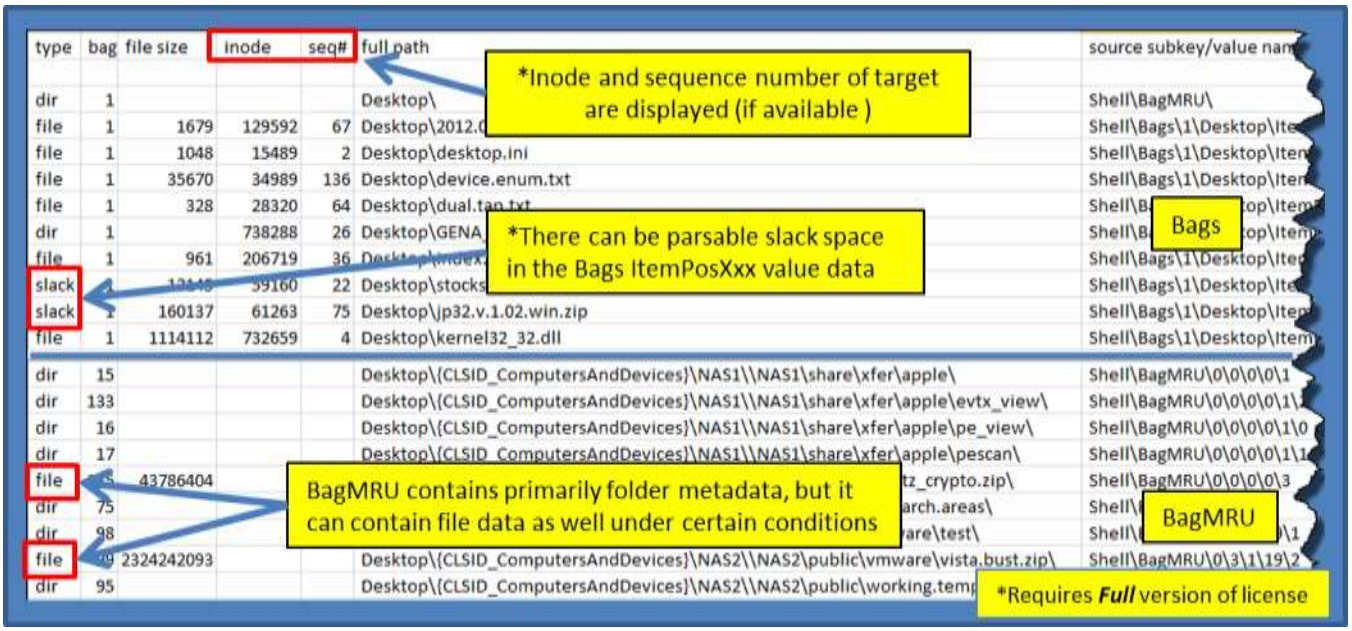


3 Results of the Parsing

In previous versions of ***sbag*** (prior to version 0.24), the *BagMRU* and *Bags* datasets were separated into independent outputs. Since the *BagMRU* data primarily represents the folders (or directories) of the files and the *Bags* dataset primarily represents the files in the folders, it made sense to integrate like *BagMRU**Bags* datasets into one output. So, starting in version 0.24, this is now the baseline output. It happens, however, that multiple datasets can still be displayed, one for the ***Shell****[BagMRU | Bags]* dataset and one for the ***ShellNoRoam****[BagMRU | Bags]* dataset.

Below is an annotated example of ***sbag's*** output rendered on a spreadsheet. The artifacts are extracted from an *ntuser.dat* file from a Windows 7 box. Each of the command line switches used in the example are explained in the *Available Options* section of this paper. Finally, all the timestamp information was truncated to highlight some of the other fields, and more specifically, some interesting items.

Example: ***sbag -hive c:\dump\ntuser.dat -base10 -inc_slack –csv > results.csv***



type	bag	file size	inode	seq#	full path	source subkey/value nam
dir	1				Desktop\	Shell\BagMRU\
file	1	1679	129592	67	Desktop\2012.0	Shell\Bags\1\Desktop\Ite
file	1	1048	15489	2	Desktop\desktop.ini	Shell\Bags\1\Desktop\Iten
file	1	35670	34989	136	Desktop\device.enum.txt	Shell\Bags\1\Desktop\Iten
file	1	328	28320	64	Desktop\dual.tap.txt	Shell\B op\Item
dir	1		738288	26	Desktop\GENA	Shell\B op\Item
file	1	961	206719	36	Desktop\index	Shell\Bags\1\Desktop\Ite
slack	1	12448	59160	22	Desktop\stocks	Shell\Bags\1\Desktop\Ite
slack	1	160137	61263	75	Desktop\jp32.v.1.02.win.zip	Shell\Bags\1\Desktop\Iten
file	1	1114112	732659	4	Desktop\kernel32_32.dll	Shell\Bags\1\Desktop\Item
dir	15				Desktop\{CLSID_ComputersAndDevices}\NAS1\\NAS1\share\xfer\apple\	Shell\BagMRU\0\0\0\0\1
dir	133				Desktop\{CLSID_ComputersAndDevices}\NAS1\\NAS1\share\xfer\apple\evtx_view\	Shell\BagMRU\0\0\0\0\1\1
dir	16				Desktop\{CLSID_ComputersAndDevices}\NAS1\\NAS1\share\xfer\apple\pe_view\	Shell\BagMRU\0\0\0\0\1\0
dir	17				Desktop\{CLSID_ComputersAndDevices}\NAS1\\NAS1\share\xfer\apple\pescan\	Shell\BagMRU\0\0\0\0\1\1
file	75	43786404			tz_crypto.zip\	Shell\BagMRU\0\0\0\0\3
dir	75				arch.areas\	Shell\
dir	98				are\test\	Shell\ \1
file	99	2324242093			Desktop\{CLSID_ComputersAndDevices}\NAS2\\NAS2\public\vmware\vista.bust.zip\	Shell\BagMRU\0\3\1\19\2
dir	95				Desktop\{CLSID_ComputersAndDevices}\NAS2\\NAS2\public\working.temp	

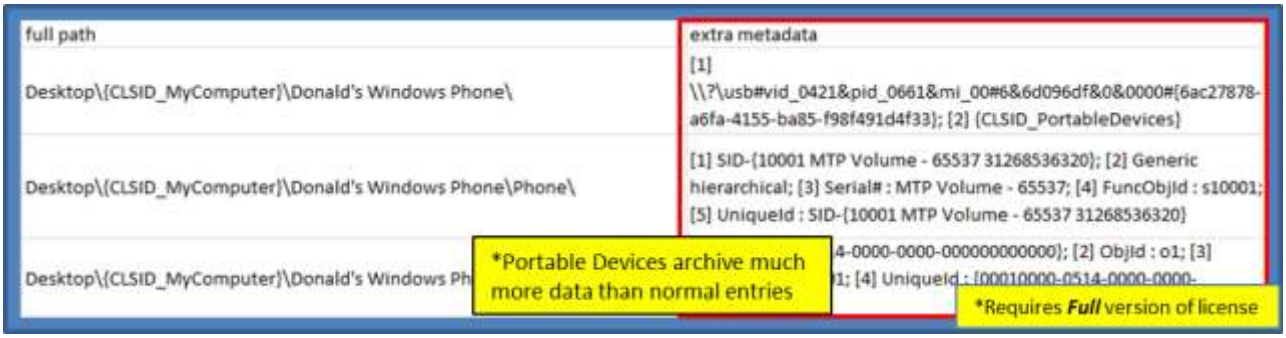
Below are some observations one should note from the above snapshot:

a. The value name *Shell\ Bags\ <bag#>\ Desktop\ ItemPos<screen resolution>* contains the metadata associated with files that were part of the Desktop. However, (not shown in the above snapshot), it can also contain metadata about *folder* data as well.
b. Since the *ItemPosXxx* registry value is a large blob of binary data, there are cases where there is sufficient cell slack space to pull out additional artifacts.
c. The *Shell\BagMRU* subkeys, while mainly containing folder metadata, can also contain *file* metadata, with complete MACB timestamps as well as size information.
d. The last column of the screenshot above shows where the data was derived from. As it happens, some of the folder/file data '*appears*' to be redundant, so this last column adds more confidence to the results presented so as to identify the origins of the data. This more easily allows one to '*hand*' parse the data to verify any of the results generated. We, in fact, use it to help verify our results as we make changes to the baseline software.
e. *Shellbag* data may contain an *inode* (MFT entry) and MFT sequence number to reference the target entry. One can use this *inode*/sequence number pair to help find the target entry, and if found, will yield additional metadata to the examiner.

sbag shows other data in the output as well, but the display is rather long; so we broke up the data further to highlight these other fields with another example. This data was taken from parsing a

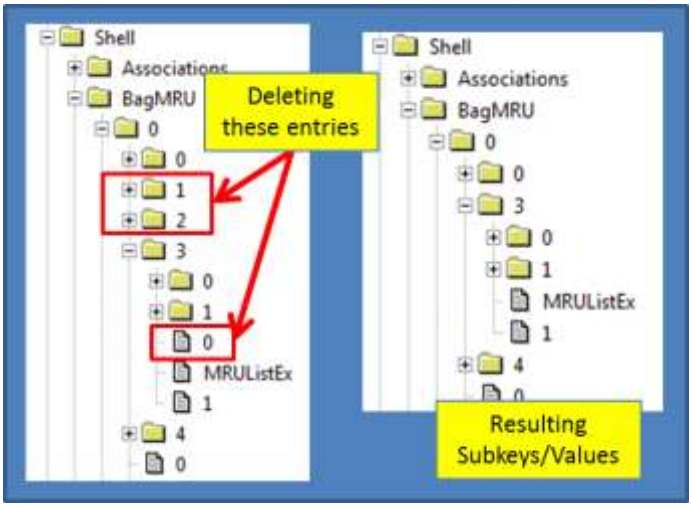
UsrClass.dat hive provided by Rob Lee and used in the SANS 408 Forensics challenge exercise. The output fields we focus on in this example are the registry key timestamps for the parent BagMRU entries and the indicator to the right of these times displays which *BagMRU* entry is the last associated with the registry time. This is known as the MRU (for Most Recently Used) entry. Also shown in this example are Bag timestamps that are related to each *BagMRU* entry. Since each BagMRU points to a Bag entry, it is useful to use these Bag timestamps to also correlate MRU times for those entries that do not have one. This gives one some additional data to help understand when this entry was last changed. One should note, in some cases, sbag provides more than one Bag timestamp (as shown below). The reason for this is that ***sbag*** extracts all the registry subkey timestamps for the parent Bag entry to identify the times this Bag entry (and thus related to the BagMRU entry) was updated as opposed to only the main one.

As the last example to show the amount of data sbag can provide, the metadata field is expanded from the previous example. Three entries are only shown to highlight the additional data portable devices store in *ShellBag* entries. From this particular entry, one can see the Vendor and Product Identifier, serial number, security identifier, etc.; useful information from an investigators perspective.

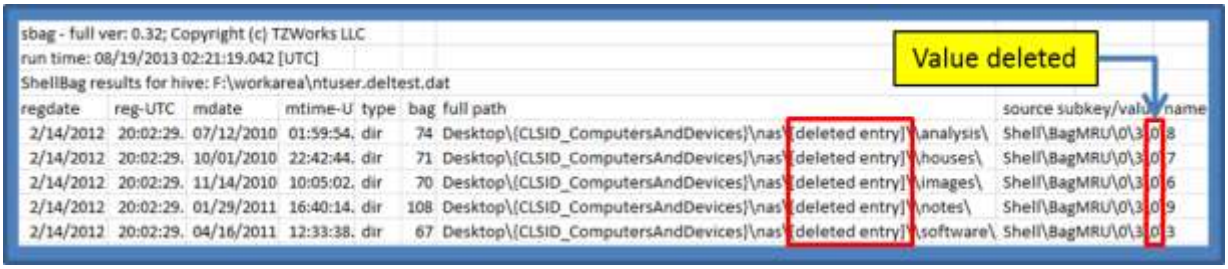


4 Results when some of the *shellbag* entries are deleted

Occasionally ***sbag*** will come across entries in the shell bag hierarchy that are missing. This most likely occurs by deleting certain entries and leaving others. Empirical data suggests that the operating system does not do this. So when this occurs, it was most likely intentionally done with anti-forensics in mind. When considering these types of cases, there are two possibilities: (a) deleting subkeys and (b) deleting values. The graphic below shows an example of this.



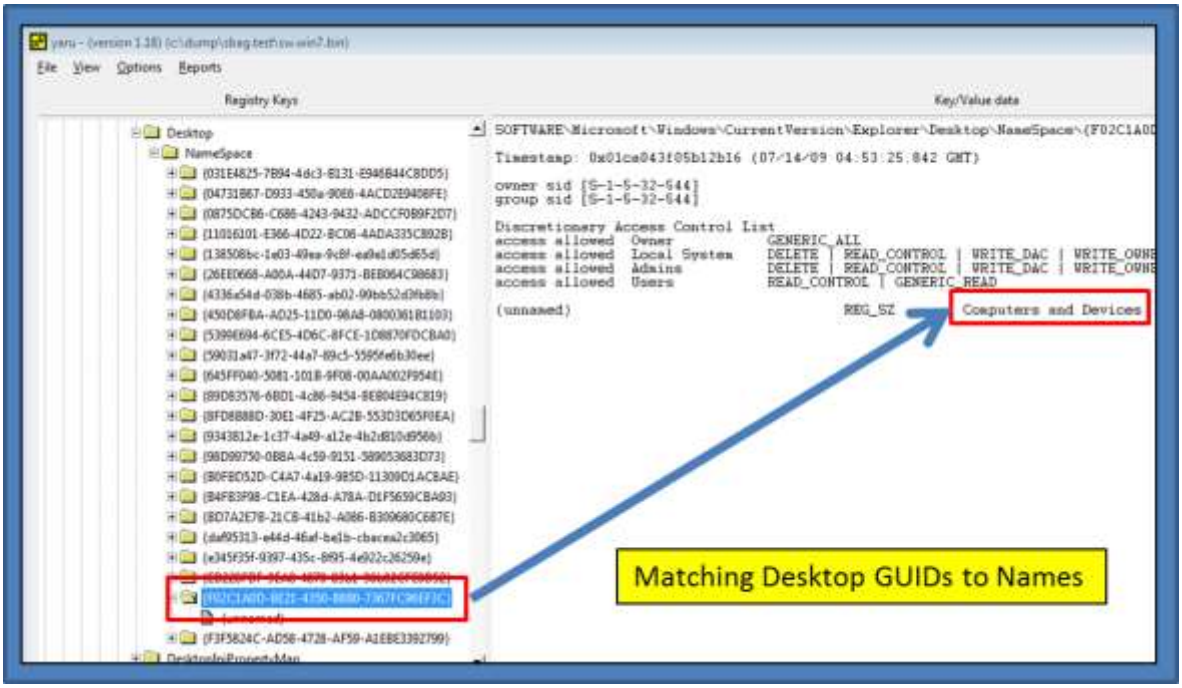
When a subkey is deleted, the rest of the entries are deleted as well. For recovery, one needs to rely on reconstructing the deleted subkeys from the registry unallocated hive space. While ***yaru*** can do this, ***sbag*** currently does not automatically do this. The second example is when one of the values is deleted. In this case, ***sbag*** can reconstruct most of the path while marking the deleted value in the path accordingly. Below is an example of the output for this second case:



5 GUIDs for Common Desktop Items

Many of the *shellbag* datasets have Globally Unique Identifiers (GUIDs) to represent common desktop items. Microsoft publishes a list of well-known GUIDs. Another location one can extract GUID/desktop item is in the desktop namespace subkeys in the software registry hive.

In the example in the previous section, there a number of entries that use the term *CLSID_ComputersAndDevices*. This name is constructed from the registry subkey *HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\Desktop\NameSpace\{GUID}.* One particular GUID is highlighted from the snapshot below along with its string definition. The GUID {f02c1a0d-be21-4350-88b0-7367fc96ef3c} from the graphic below equates to the name "Computers and Devices." Thus, when ***sbag*** finds a GUID, such as the one above, it uses this GUID/name relationship to generate the name *CLSID_ComputerAndDevices* when rendering the output.



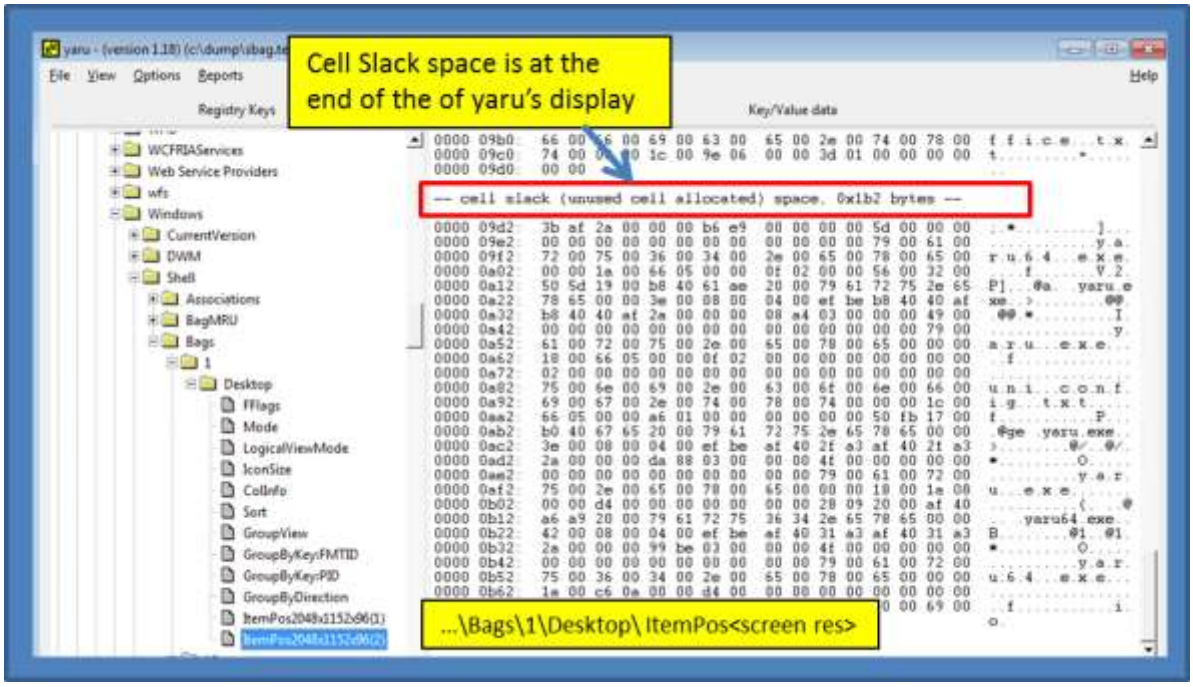
Other names representing GUIDs that one will see ***sbag*** displaying regularly are listed below. Many of these can be resolved in one of the namespaces in the Software registry hive.

CLSID_MyComputer = {0d04fe0-3aea-1069-a2d8-08002b30309d}
CLSID_RecycleBin = {645ff040-5081-101b-9f08-00aa002f954e}
CLSID_ControlPanel = {21ec2020-3aea-1069-a2dd-08002b30309d} & { 26ee0668-a00a-44d7-9371-beb064c98683}
CLSID_MyNetworkPlaces = {208d2c60-3aea-1069-a2d7-08002b30309d}
CLSID_MyDocuments = {450d8fba-ad25-11d0-98a8-0800361b1103}
CLSID_UserLibraries = {031E4825-7B94-4dc3-B131-E946B44C8DD5}
…

6 Cell Slack Artifacts in Bags *ItemPos* Data

For cases where there is sufficient slack space available in the registry cell value data, ***sbag*** will try to parse this data into some intelligible output. Since this option is still experimental, it has been added as a separate option one can invoke. This allows one to isolate the option, should it exhibit any instability, until sufficient testing has been done. To use this option, use the ***–inc_slack*** switch.

For those wishing to analyze cell slack space in a hex dump view, one can use our ***yaru*** [7] utility. Just navigate to a similar registry value that is shown below, and scroll to the end of the data output. If there is any slack space, it will be displayed at the end. The results of processing the slack space below was shown in the screenshot displayed on section 3 above.



7 Using Inode Information to Analyze an Entry

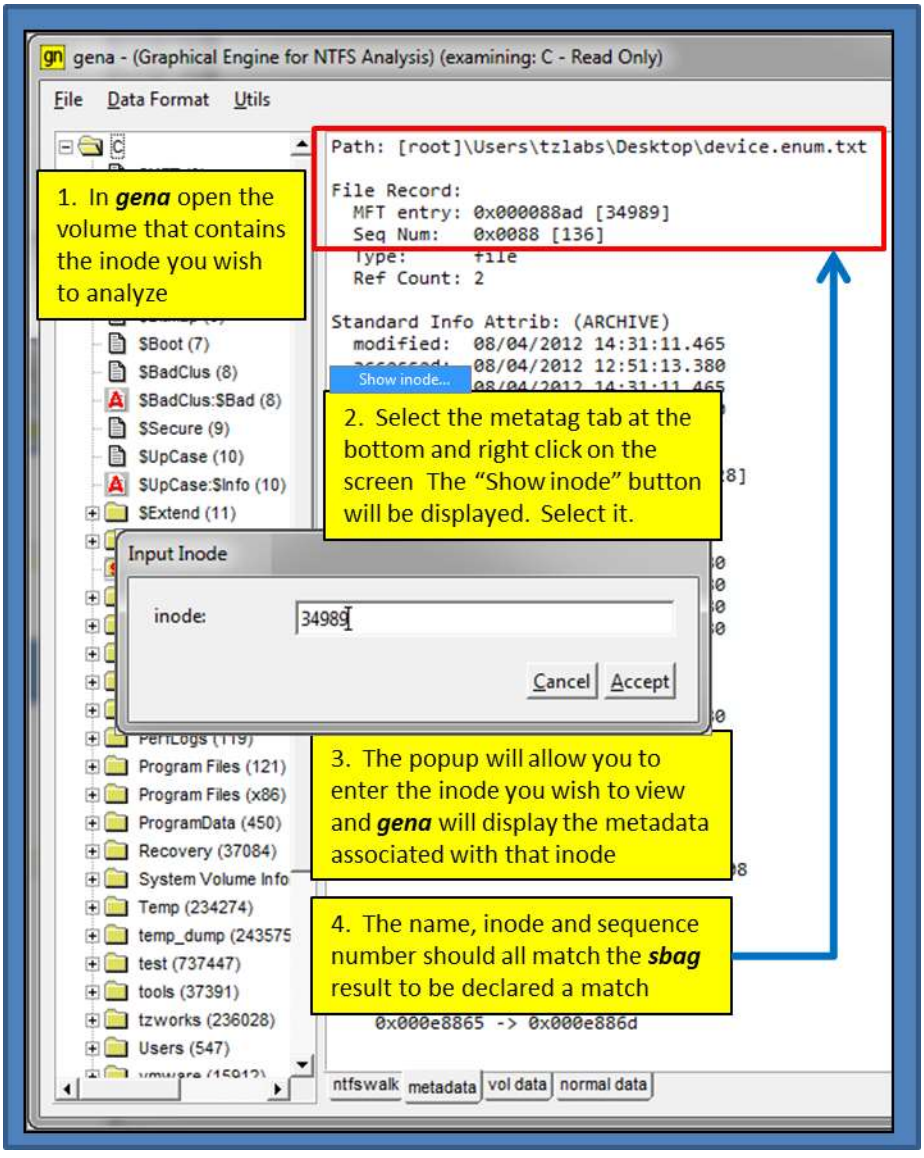
To do this, we will make use of the ***gena*** tool to help provide insight to specific inode's. Note, one could have used ***ntfswalk***, ***ntfsdir***, etc., or any tool that allows for inode evaluation. For this example, we will use the results shown on screen shot in section 3 above. Only a portion of this information is shown below, so we can focus on the entry that is highlighted in red.

type	bag	file size	inode	seq#	full path
					Desktop\
				67	Desktop\2012.08.ion
				2	Desktop\desktop.ini
file	1	35670	34989	136	Desktop\device.enum.txt
file	1	328	28320	64	Desktop\dual.tap.txt

Going further with the inode data to analyze the specific file

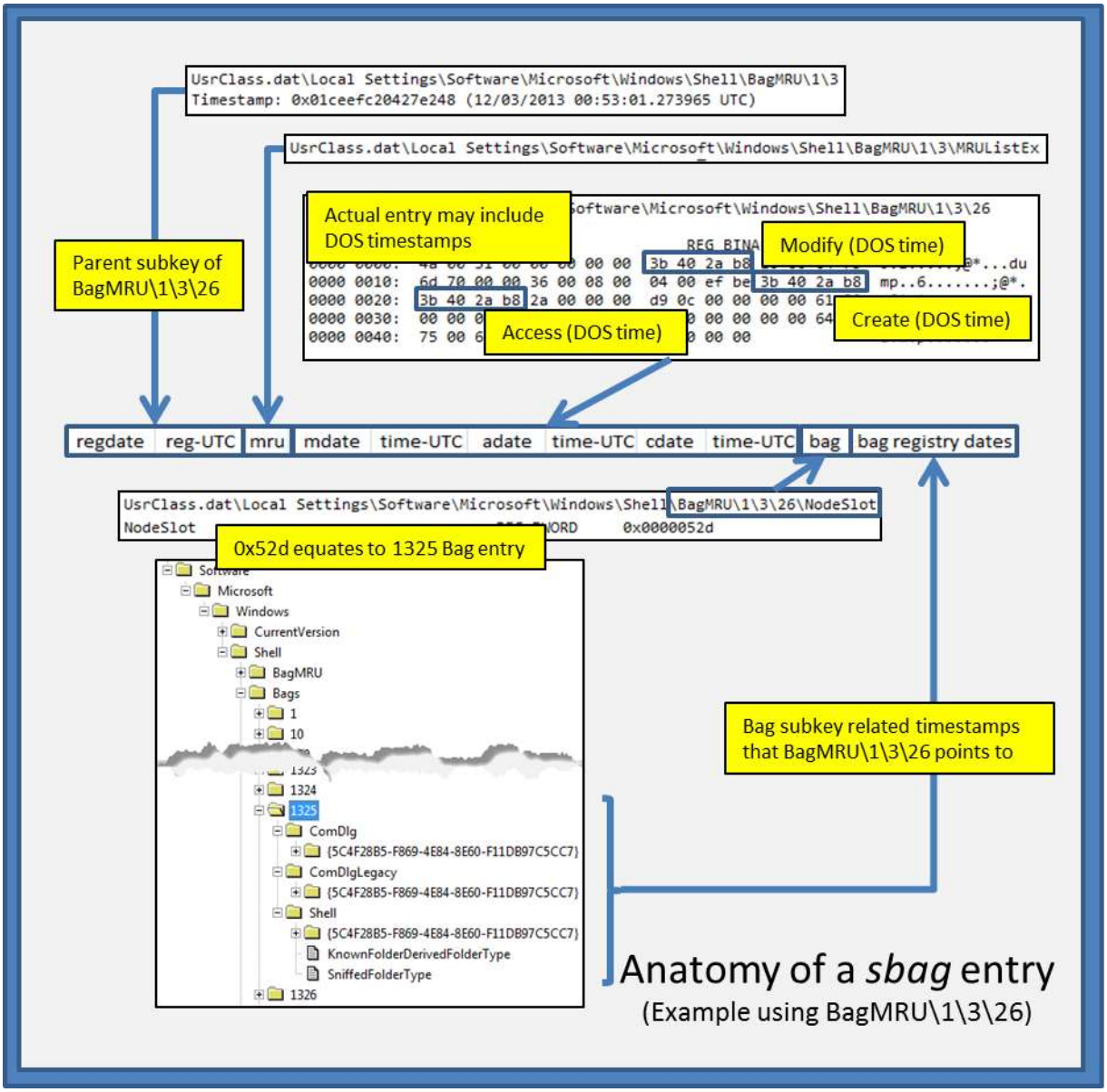
By using the inode and sequence number, one can confidently find the entry in question on a volume. One must, of course, identify the volume in question, since a desktop item shown in the ***sbag*** results could have been sourced anywhere. For this case, to make it easy, we selected an entry that came from the 'C' volume.

When using ***gena*** to analyze this entry, there are really a few steps to get to the entry desired. Each of these steps is enumerated in the graphic below:



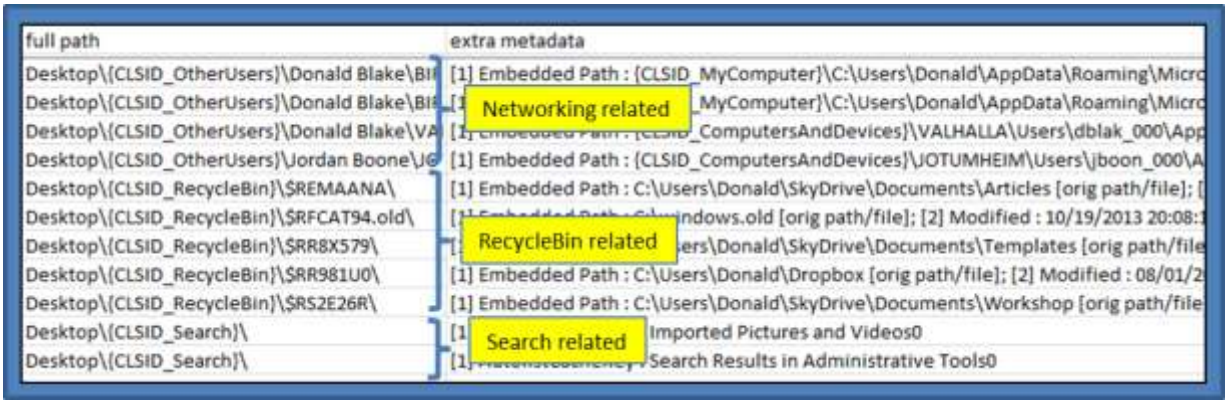
8 Anatomy of a *sbag* Entry

Each entry, ***sbag*** shows data from various sources within the *BagMRU* and *Bags* subkeys. Parsing this manually is confusing and prone to errors. To show where all data is taken from, an example is taken by looking at the *Shell\BagMRU\1\3\26* entry and its corresponding sources of data. Each source of data is annotated with the actual registry data used to populate the ***sbag*** entry. This includes: (a) time from the parent subkey (Shell\BagMRU\1\3\26), (b) *MRUListEx* entry that is a child of the parent directory, (c) DOS timestamps embedded in the *shellbag* data in the target entry, (d) *NodeSlot* data that identifies the companion *Bag* entry, and (e) the corresponding *Bag* entry subkey timestamps.



8.1 Extra Metadata

Aside from the normal parameters extracted, ***sbag*** will also try to parse any other data it finds. This data is typically additional properties specific to that entry and can details such as dates, more path information, icons used, etc. Any additional properties that are parsed are displayed to the user in the column titled "*extra metadata*". This field is currently just a catchall for any extra information. This field can get populated by extra data found in entries from: (a) RecycleBin, (b) cellphone, (c) networking, (d) Music, (e) Pictures, (f) Searches, etc. By looking at the 'extra metadata' one can see if this entry has any unique properties. Below is an example of grouping a few of these entries.



8.2 Timestamps

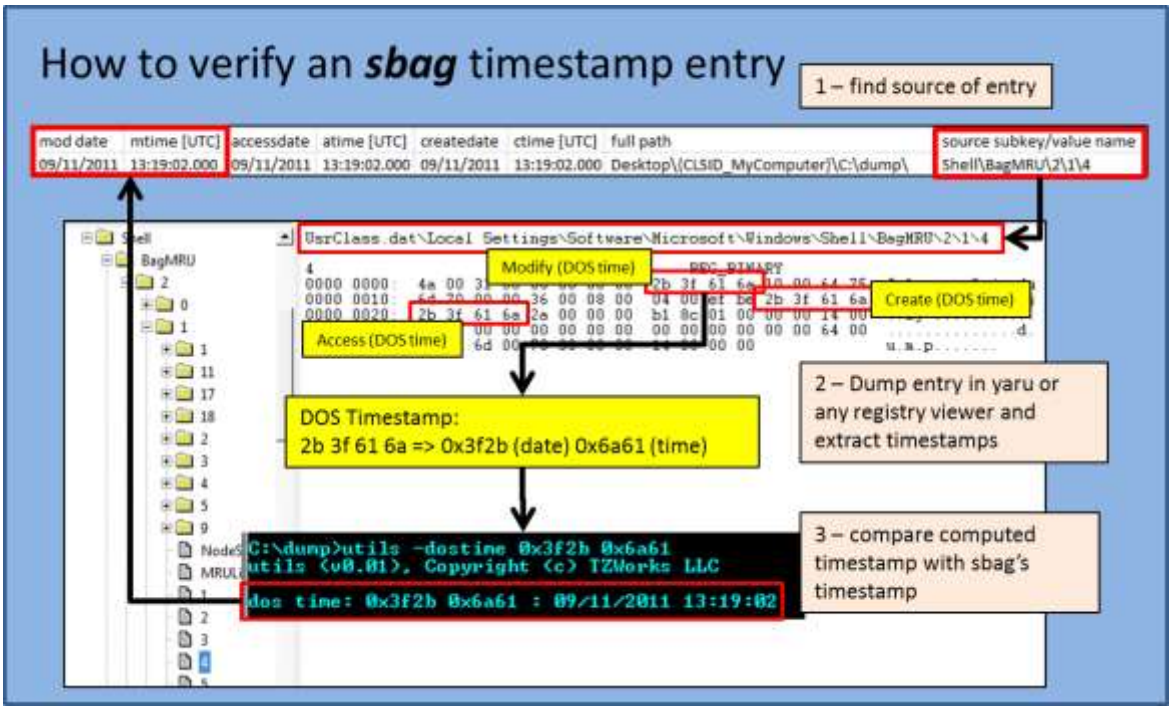
When parsing the last Item ID node for a particular entry, if there are DOS time stamps present, ***sbag*** will populate the fields: modify date (mdate/time), access date (adate/time) and create date (cdate/time). Obviously, if none are present, these fields are blank. In some cases, however, ***sbag*** will include additional time stamps that get reported in the '*extra metadata'* field. In these cases, ***sbag*** found both DOS time stamps as well as other time stamp data embedded in one of the property structures for that node. In some cases, the DOS time stamps will not be present, but there will be timestamps available as a property.

9 Timestamp verification

For ***sbag*** algorithm updates, we internally perform regression testing as a normal course of action to verify the entries are valid. Likewise, it is highly recommended that any user of our tools do some sort of integrity validation to ensure the data reported is accurate. Below is a quick way of one way to verify the timestamps reported by ***sbag*** reflect the raw data.

One starts out by identifying where the source of an entry came from (shown as #1 in the diagram). This can be viewed in the last column of the ***sbag*** output. Next, one can use any registry viewer to extract the binary data from the appropriate cell value (shown as #2 in the diagram). For the example below,

we used ***yaru*** to extract and review the binary data. The timestamps embedded are DOS based (versus FILETIME based), and thus, are four byte values. After locating the three DOS timestamps, one converts these timestamps into a readable form. Step #3 below shows a multipurpose utility we use to convert between various time formats, however, any trusted time conversion tool that is available online will suffice.



10 Available Options

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

Option	Description
-hive	Use this option to specify which hive to process artifacts from. Syntax is ***-hive <hive file>***
-vss	Experimental. This option allows one to point to a Volume Shadow copy and process any user hives in the standard users' directories. Syntax is ***-vss <index of volume shadow copy>***. Only applies to Windows Vista, Win7, Win8 and beyond. Does not apply to Windows XP.
-partition	Experimental. This option allows one to point to a drive letter and process any user hives in the standard users' directories. Syntax is ***-partition <drive letter>***
-livehives	Switch to enumerate the user hives on the current system. Since this option only makes sense when running ***sbag*** on a Windows box, this option will not show up when running on Linux or Mac OS-X.
-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
-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>***.
-userstats	Switch tells ***sbag*** to try to extract the user account name from the

	ntuser.dat hive and populate it in the data that is outputted.
-pipe	Used to pipe files into the tool via STDIN (standard input). Each file passed in is parsed in sequence.
-enumdir	Experimental. Used to process files within a folder and/or subfolders. Each file is parsed in sequence. The syntax is ***-enumdir <folder> -num_subdirs <#>***.
-filter	Filters data passed in via STDIN via the ***-pipe*** or ***-enumdir*** options. The syntax is ***-filter <"*.ext | *partialname* | ...">***. The wildcard character '*' is restricted to either before the name or after the name.
-inc_slack	Switch to extract and parse any slack data in the Bags ItemPosXxx value data.
-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.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
-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.

11 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.

11.1 *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.

12 References

1. MiTec Registry Analyzer, by Allan S Hay, 12/2004
2. Shell BAG Format Analysis, by Yogesh Khatri
3. Using shellbag information to reconstruct user activities, Yuandong Zhu, Pavel Gladyshev, Joshua James, Centre for Cybercrime Investigation, University College Dublin, Belfield, Dublin 4, Ireland, DFRWS 2009
4. SANs Institute. Forensics 408 course (Jan 2010)
5. SleuthKit Body-file format, http://wki.sleuthkit.org
6. Log2timeline CSV format, http://log2timeline.net/
7. yaru - Yet Another Registry Utility, www.tzworks.com