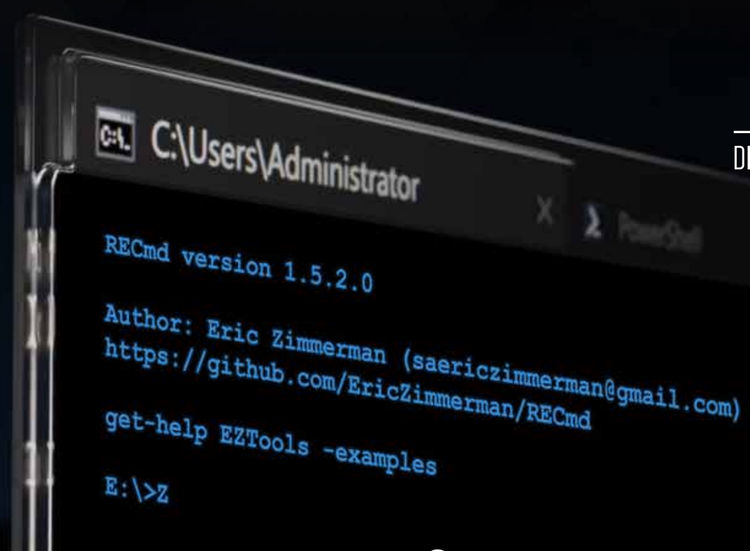



SANS DFIR
DIGITAL FORENSICS & INCIDENT RESPONSE

# Results in Seconds at the Command Line

DFPS_Command-Line_v1.2_01-21

## DIGITAL FORENSICS & INCIDENT RESPONSE

### Operating System & Device In-Depth

**FOR308** — Digital Forensics Essentials

**FOR498** — Battlefield Forensics & Data Acquisition
GBFA

**FOR500** — Windows Forensic Analysis
GCFE

**FOR518** — Mac and iOS Forensic Analysis and Incident Response

**FOR585** — Smartphone Forensic Analysis In-Depth
GASF

### Incident Response & Threat Hunting

**FOR508** — Advanced Incident Response, Threat Hunting, and Digital Forensics
GCFA

**FOR572** — Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response
GNFA

**FOR578** — Cyber Threat Intelligence
GCTI

**FOR610** — Reverse-Engineering Malware: Malware Analysis Tools and Techniques
GREM

**SEC504** — Hacker Tools, Techniques, Exploits, and Incident Handling
GCIH

digital-forensics.sans.org

# Forensics the EZ Way:

**With the wealth of data stored on Windows computers it is often difficult to know where to start. If you encounter a sizable hard drive, it could be hours or even days before you're ready to even start your investigation, never mind reporting the results. Using the EZ tools provides scriptable, scalable, and repeatable results with astonishing speed and accuracy. Go from one investigation a week to several per day. This type of performance is common with the command line versions of EZ Tools. This poster will show you how.**

Eric Zimmerman's TOOLS

### Common CLI Options & Switches

Short options (single letter) are prefixed with a single dash.
Long options are prefixed with two dashes.

| Options | Definition |
|---|---|
| -d | Dir to process |
| -f | File to process |
| -q | Quiet mode |
| --dt | Custom date/Time format |
| --mp | Higher precision timestamps are displayed and will also be reflected in any exported data |
| --csv --json --html | Data can be exported to several formats. You can request multiple formats at the same time. |
| --debug | Shows debug info during tool execution (more info) |
| --trace | Shows trace info during tool execution (most info) can be run with debug (--debug --trace) |
| --sync | Sync updates from GitHub for KAPE targets & module updates. For evtxecmd map updates |
| -vss | Process Volume Shadow Copies – Supported in EvtxECmd, MFTECmd, PECmd,and RECmd |

## AppCompatCacheParser – Shimcache Parser

### Type of Artifact

Application Compatibility Cache (also known as Shimcache) is part of the Windows capability to provide backwards compatibility for programs meant to run on older versions of Windows. When an executable is found by Windows, the operating system determines how best to run the program. As a byproduct of this, the AppCompatCache stores information about those executables. AppCompatCache can be leveraged to assist forensic investigators in determining what executables were run on Windows.

### Basic Usage

`AppCompatCacheParser` takes the SYSTEM registry hive as input and interprets the data stored therein. The SYSTEM hive must be specified using the -f parameter to indicate that the command is to be run over that registry hive only.

In the example command below, `AppCompatCacheParser` is being run against a SYSTEM registry hive stored on an evidence file mounted as a disk (`E:`). Output is stored on the G: drive to the "AppCompatCache" folder. The `AppCompatCacheParser` application will create an output file (CSV in this case) with the date and time that the AppCompatCacheParser was executed and in the detected version operating system, in the file name.

```
AppCompatCacheParser.exe -f E:\Windows\System32\config\SYSTEM
--csv G:\AppCompatCache
```

### Key Data Returned

Processed Shimcache data in CSV, XML, or JSON format is available. The columns of most significance are typically the "Path" (the location and name of the executable), "LastModifiedTimeUTC" (the last written time of the executable) and "Executed" (whether the executable was run). The most common mistake made by forensicators is that they'll assume that the LastModifiedTimeUTC value refers to the execution of the file. Don't fall into this trap!

### Advanced Usage

**PRO TIP:** Watch for changes at the start of the "Path". Anything that shows "`SYSVOL`" ran from the host's OS volume. Other volumes will be recorded by their drive letter.

| Path | Last Modified Time UTC | Executed |
|---|---|---|
| SYSVOL\Windows\System32\notepad.exe | 8/22/2019 11:00:12 | Yes |
| E:\TACTICAL Subject\f-response-tacsub.exe | 8/12/2019 19:21:00 | Yes |

**PRO TIP:** As a file's last written time does not change when a file is moved, renamed or copied, it may be possible to track the same executable across a single or even multiple systems, as a new entry will be created in the AppCompatCache when the file is executed from a different location or with a different name. The table below shows the same executable being run in different scenarios. We know they are all the same executable because they share the same last written time.

| Path | Last Modified Time UTC | Executed |
|---|---|---|
| SYSVOL\Windows\System32\spinlock.exe | 10/23/2019 14:27:18 | Yes |
| SYSVOL\Users\SRogers\AppData\Local\Temp\spinlock.exe | 10/23/2019 14:27:18 | Yes |
| SYSVOL\Windows\prune.exe | 10/23/2019 14:27:18 | Yes |

## PECmd – Prefetch Parser

### Type of Artifact

Prefetch is one source of evidence of execution or evidence of a program being run on a system. Prefetch files are created in the `C:\Windows\Prefetch` folder when a program is run from a specific location. If that program is run from more than one location, there will a separate prefetch file created for each location from which the program ran. Prefetch files are not automatically deleted if the related program is deleted and therefore can be a source of historical information.

The creation date of the prefetch files is generally the first time that the program was attempted to be run from a particular location. This statement is intentionally vague because 1. Programs that do not successfully run can still create a prefetch file and 2. The prefetch directory is limited to 128 files so files are overwritten. A prefetch file can be created for a program that has previously run and it's prefetch file overwritten. This time can be off or delayed by approximately 10 seconds.

### Basic Usage

Process a single Prefetch files and send results to screen

```
PECmd.exe -f C:\Windows\Prefetch\CMD.EXE-8E75B5BB.pf
```

Process a directory of Prefetch files and send results to a CSV file named prefetch.csv. The `--csvf` allows you to provide the name of the prefetch output csv.

```
PECmd.exe -d C:\Windows\Prefetch\ -q --csv G:\Prefetch --csvf
prefetch.csv
```

Process a directory of Prefetch files, including VSS, and send the results to a CSV file named prefetch.csv and higher precision timestamps

```
PECmd.exe -d C:\Windows\Prefetch\ -q --csv G:\Prefetch --csvf
prefetch.csv --vss --mp
```

### Key Data Returned

`PECmd`, in csv mode, will output two CSV files, one of which is a timeline. The Timeline csv will have"_Timeline in the file name. The main Prefetch ouptut file will contain important information such as:

- Executable name and full path from which it was executed
- Volume name and serial number from which the program ran
- Run Count – the number of time that the program was run, from that location
- Timestamps (UTC) for the last eight executions
- Volumes,Files and directories accessed during execution.

***Console Output Color:***

Items in Red: Any path that has the strings "temp" or "tmp" in them. Also, any key word that you included on the command line using the "`-k`" option and appears in the path, will be shown in red.

Items in Yellow: Executables that are referenced in the files section are highlighted in yellow.

### Advanced Usage

**KEYWORDS:** You can provide a comma separated list of keywords on the command line. Volumes, directories, and files accessed by the program, that are responsive to these keywords, will be shown in red on the display.

```
PECmd.exe -d C:\Windows\Prefetch\ -q --csv G:\Prefetch --csvf
prefetch.csv -k "system32, downloads, fonts"
```

**PRO TIP:** `PECmd` can extract and process Prefetch files from Volume Shadow Copies by using the "`--vss`" option. This will process Prefetch from ALL Volume Shadow Copies. The output files will be separated by individual VSS numbers.

```
PECmd.exe -d C:\Windows\Prefetch\ -q --csv G:\Prefetch --csvf
prefetch.csv --vss
```

Note: To decompress the new compressed prefetch files created with Windows 10, you must run PECmd on Windows 8 or later.

## VSCMount – Volume Shadow Copy Mounter

### Type of Artifact

Volume Shadow Copies are created periodically to capture the previous state of a system. This means that deleted and wiped files, or even older versions of a file or folder, can be recovered from volume shadow copies. In order to recover such data a volume shadow copy must be mounted. VSCMount allows an investigator to mount each volume shadow copy.

### Basic Usage

Before running the `VSCMount tool`, an evidence file must itself be mounted as a physical drive. Arsenal Image Mounter is the ideal tool for this task. Open Arsenal Imager Mounter and click on "Mount disk image". Then select the evidence file to mount. Ensure that the option to "Write temporary disk device" is enabled and click "OK".

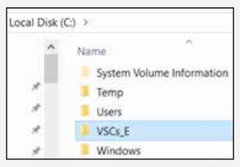
Once mounted, note the drive letter. In the example below it is drive letter E.


Open an Administrator PowerShell window and run `VSCMount`. In the example command below, the "`--dl`" switch stands for "drive letter". This is the drive letter from the evidence file mounted above. The "`--mp`" switch stands for "map point". In this example, the drive letter is "E". This is the location where `VSCMount` will create the links to all of the volume shadow copies found on the mounted evidence. In this instance, the volume shadow copies will be mapped to "`C:\VSCs`".

```
.\VSCMount.exe --dl E --mp C:\VSCs
```

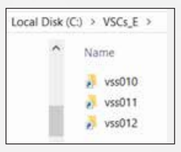
### Key Data Returned

When run, `VSCMount` counts the number of Volume Shadow Copies on the mounted image and maps each one to the target directory. From the example command given above, `VSCMount` found and mapped three volume shadow copies.




Note that `VSCMount` has appended the "map point" with the drive letter.

Inside the map point, there are three mapped volume shadow copies from the mounted E drive. Each of these can be expanded and viewed as needed.




### Advanced Usage

**PRO TIP:** Looking at the mapped Volume Shadow Copies, it isn't immediately clear as to when they were created. Adding the "`--ud`" switch to the command adds the creation date of each mapped Volume Shadow Copy, as shown in the example below:

```
.\VSCMount.exe --dl E --mp C:\VSCs --ud
```


## bstrings – Extract Text From Binary Files

### Type of Artifact

Bstrings can be used to search any type of file for potentially valuable information.

### Basic Usage

```
bstrings.exe -f <file>
```

| Option/Switch | Use | Example |
|---|---|---|
| --ls | Search for string | bstrings -f suspect.exe --ls password |
| --lr | Search with regular expression | bstrings -f suspect.exe --ls (ntos\|win32k) |
| --p | List builtin regular expressions | bstrings -p |
| --lr XX | The XX represents a builtin regex | bstrings -f suspect.exe --lr ipv4 |
| --fr | Read file containing regex's to use in search | bstrings -f suspect.exe -fr DFIR_RegExs.txt |
| -h | List all options | bstrings -h |
| --cp | Use a different ANSI code page | bstrings -f Powershell.evtx --ls download --cp 1201 |

note: Windows Event Log require the 1201 specific code page for bstrings to find the search string

Interesting options and switches:

```
bstrings.exe -f <file> --ls "password"
```

Use the `-x` and `-m` switches to set maximum and minimum string lengths.

Use `--off` to show the offset for each search hit.

### Advanced Usage

`--lr` Regular Expression searches `bstrings` and also contains over a dozen built-in regular expression patterns for things like credit card numbers, social security numbers, IP addresses, email addresses, and more.

`-p` shows a list of built-in regular expressions. When using a built-in expression, use the value in the Name column. For example, to look for email addresses, use this command:

```
bstrings.exe -f <some file> --lr email
```

`bstrings` also allows searching for several strings or regular expressions at once using the `--fr` and `--fs` switches.

In addition to Unicode strings, `bstrings` looks for strings encoded using Western (1252) code page. Use the `--cp` switch to search in any other code page supported by .net.

A full listing of available code pages is available at **https://goo.gl/ig6DxW**

## EvtxECmd – Windows Event Log Parser

### Type of Artifact

There can be hundreds of Event Logs in the evtx folder, some aimed at system-wide events like `Security.evtx`, `System.evtx` and `Application.evtx`. There can be many others that record information in a much more targeted fashion. All Event Logs are stored in the same format on a Windows computer but the actual data elements collected varies. It is this variation of data elements that makes correlation of Event Logs a challenge. This is where `EvtxECmd` shines. All event records are normalized across all event types and across all Event Logs file types!

The `EvtxECmd` parser has standardized CSV, XML, and JSON output. It also has custom maps and locked file support, and it's unbelievably fast. EvtxECmd has a unique feature, "Maps," that allows for the normalized output format.

Event Log Location: Event Logs for Windows Vista or later are found in `%systemroot%\System32\winevt\logs`

Although you may preserve all the logs, you probably would not parse and analyze them all. The same statement can be made for all the `Event _ IDs` in the Event Logs files. `EvtxECmd` makes the selective processing of Event Logs easier and scriptable.

### Basic Usage

Recursively parsing a directory of event logs is probably the most efficient way to use `EvtxECmd`. To parse a directory, copy Event Logs to a temporary directory and use the `-d` option. Additionally, use the `--inc` option to only include specific `Event _ IDs` in the processing.

You have extracted the Event Log to a folder named `e:\evtx\logs` and now you want to process all those logs in a single command.

```
EvtxECmd.exe -d E:\evtx\logs --csv G:\evtx\out --csvf evtxecmd_
out.csv
```

Process all event logs and only include event_id specified by the `--inc` option

```
EvtxECmd.exe -d E:\evtx\logs --csv G:\evtx\out --csvf evtxecmd_
out.csv --inc 4624,4625,4634,4647,4672
```

Exclude specific event_id's by using the `-exc` option

```
EvtxECmd.exe -d E:\evtx\logs --csv G:\evtx\out --csvf evtxecmd_
out.csv --exc 4656,4660,4663
```

### Key Data Returned

Processed Event Logs are in a standardized CSV, XML, or JSON format. Output normalization is accomplished through the use of Event Log "maps". Maps provide specific handling of a single combination of Event Log and Event ID. Events without maps are still processed, but output format will vary. The normalized Event Log output makes it possible to analyze many different types of Event Logs in a single view. Timeline Explorer is perfect for this analysis.

### Advanced Usage

**PRO TIP:** Process only the Event Logs and Event IDs that are relevant to your case.

Check out this PowerShell script that copies out the relevant Event Logs and processes only specific Event IDs (your list of relevant logs and Event IDs may vary).
**https://for500.com/evtx2process**

## SBECmd – Shellbags Explorer

### Type of Artifact

The Shellbags artifact is truly amazing. Every time Windows Explorer interacts with a folder, an entry is created in the computer's Shellbags. Folders also include other "Explorer Like" items like the Control Panel, zip files, ISOs, and mounted encrypted containers. Shellbags entries are not prepopulated based upon the systems folder structure, they are only created with the user interaction occurs. The simple existence of a directory in Shellbags is evidence the specific user account once interacted with that folder. Shellbags entries are likely to persist even when the original directories, files, and physical devices have been removed from the system and due to this, can serve as a "history" of sorts of data that was previously on a system but may have since been removed.

ShellBags are a set of Windows Registry keys located in `NTUser.dat` and `USRClass.dat` Registry hives (primarily `USRClass.dat`) that maintain view, icon, position, size (and other attributes) of folders when using Windows Explorer. We used to say the Shellbags tracked folders that a user opened. Our understanding of the artifact has subsequently grown, and we now know that simply bringing a folder into "focus" in Windows Explorer can create and populate the Shellbags registry key. Moreover, different levels of interaction can populate more or less of the Shellbags fields. At a high level, opened == accessed == interacted.

Surely Shellbags was not created for the purpose of tracking use folder access. The official statement is that they were created to enhance the user experience by "remembering" the Windows Explorer view settings, on a folder by folder basis. When a user returns to a folder, they see that layout, like medium icons sorted by most recently modified.

### Basic Usage

`SBECmd` uses `-d` for a directory to recursively process user registry hives. There is no `-f` option for `SBECmd`.

To process a single user's ShellBags data, use the following command:

```
SBECmd.exe -d E:\Users\nromanoff --csv G:\temp\sbe_out
```

**PRO TIP:** If you need to process several users ShellBags data, you might consider exporting their data first and then processing just folder containing the exported data. This is a performance decision. Recursively processing many user folder and be very slow.

To process all Users in the Users folder, use the following command.

```
SSBECmd.exe -d E:\Users --csv G:\tmp\sbe_out
```

### Key Data Returned

CSV with the full path of folder, MACB times for target folders and first and last folder interaction times. The Bag Path, Slot, Node Slot, and MRU position for each entry are also shown. These can initially be confusing to decipher in table form. Using the GUI verion of ShellBags Explorer to see the table view translated in a hierarchal tree format can be very useful.

**Timestamps Shown in SBECmd output:**

Because of the nature of how registry key timestamps have only a single last update value for each key, the hierarchal data in the BagMRU registry key can become stale. This means that there may be a value in the key but it could be outdated. Therefore if `SBECmd` is not positive that a date is current and accurate, that date will not be shown in the output. This why you will often see that an entry has a `Last Interacted Timestamp` an no `First Interacted Timestamp`. The `First Interacted Timestamp` is stale and can't be relied upon.

You will also notice that SBECmd will only show Last Interacted Timestamps for MRU values.

### Advanced Usage

**PRO TIP:** `SBECmd` can pull data from a live system. This make for a great learning and testing feature. Pull some baseline Shellbags data, run a test like navigating into a folder, pull the data again and compare. See what you own activity does to the Shellbags data.

# RBCmd – Recycle Bin Artifact Parser

## Type of Artifact

Windows stores information relating to user deletions on a per user basis in the Recycle Bin. Windows XP used a file named "INFO2" to track the deletions. This file included the original location and time that each file was deleted. That behavior changed in Windows Vista when each deleted file was tracked on its own. Now, when a file is deleted, it is renamed. For example, if cat.jpg was deleted, the deleted file would have a name such as $R7YQ28P.jpg. The $R prefix means that it contains the content (Resource) of the original file. In addition to the $R file, a new corresponding $I (Information) file is created in the Recycle Bin. So every deleted file has both a $R and $I file with a matching random string for the rest of the file name. The $I file contains the information about the original location of the file and the date and time of deletion. RBCmd takes this data and presents it in a human-readable format.

## Basic Usage

In the example command below, RBCmd is being run against an INFO2 file stored on an evidence file mounted as a disk (E:). When running this command the output is shown in the window running the command (command line window or PowerShell). Note that because the INFO2 file may contain information about several deleted items, it may be best served to output to a CSV for review (see third example below).

```
RBCmd.exe -f E:\RECYCLE\S-1-5-21-3001495921-
1769015868-3887507880-1001\INFO2
```

In the next example, RBCmd is being run against a single $I (information) file on a mounted drive (E:). The output is displayed in the window where the command was run.

```
RBCmd.exe -f E:\$Recycle.Bin\S-1-5-21-718126207-1171771683-
1750804747-1001\$I7YQ28P.jpg
```

```
Source file: .\$IG1VEXX.xls

Version: 1 (Pre-Windows 10)
File size: 16384 (16KB)
File name: C:\Users\Donald\SkyDrive\Documents\WACC Calc Spreadsheet -SECRET.xls
Deleted on: 2013-10-21 18:32:52.5320000
```

In the final example, RBCmd is being run against the parent folder of the $I file above, thereby parsing all of the $I files. This time, the output is stored in a CSV stored in G:\RBFiles with the date and time in the file name. Use of the -q switch prevents all of the output from being sent to the window, making processing faster.

```
RBCmd.exe -d F:\$Recycle.Bin\S-1-5-21-718126207-1171771683-
1750804747-1001 --csv G:\RBFiles -q
```

| Source Name | Deleted On | File Name | File Size |
|---|---|---|---|
| $IG1VEXX.xls | 2013-10-21 18:32:52.5320000 | C:\Users\Donald\SkyDrive\Documents\WACC Calc Spreadsheet -SECRET.xls | 16384 |
| $ISFWP49.xls | 2013-10-21 18:32:52.5550000 | C:\Users\Donald\SkyDrive\Documents\WACC Calc Spreadsheet -SECRET-Bifrost.xls | 16384 |

## Key Data Returned

Processed Recycle Bin data is either output to the screen (if no output file is specified) or in a standardized CSV, XML, or JSON. The screenshot below shows an example of the output when run against a single file. The source file is shown, as is the file size, original file name and location and date of deletion.

## Advanced Usage

**PRO TIP:** Running RBCmd on a mounted drive will work, but remember that when doing so, Windows does not see deleted files, so RBCmd won't pick them up. It is often worth extracting deleted $I files using another tool and then running RBCmd over those recovered files.

# AmcacheParser – Amcache Parser

## Type of Artifact

Amcache is part of the Application Experience Service in Windows. The Application Experience Service monitors executables and determines if those programs require updating when run. As a byproduct of this, the Amcache stores information about those executables. AmcacheParser can be leveraged to assist forensic investigators in determining what executables were run on Windows, when they were run and provide a SHA-1 hash of the executables in order to track the same executables across assets.

## Basic Usage

AmcacheParser takes the Amcache.hve registry hive as input and interprets the data stored therein.

In the example command below, AmcacheParser is being run against an Amcache.hve registry hive stored on an evidence file mounted as a disk (E:). Output is stored on the G: drive to the "Amcache" folder. The AmcacheParser application will create an output file (CSV in this case) with the date and time in the file name.

```
AmcacheParser.exe -f E:\Windows\AppCompat\Programs\Amcache.hve
--csv G:\Amcache
```

## Key Data Returned

Processed Amcache data in a standardized CSV, XML, or JSON format is available. The columns of most significance are typically the "FileIDLastWriteTimestamp" (the first time the executable was run), "SHA1" (the SHA-1 hash of the file being executed) and FullPath (the location and name of the executable ran). Other data of potential interest include the Volume ID (used to determine from which volume the executable was run), MFT Entry number and Sequence numbers (used to determine if the executable was run from an NTFS volume) and information about the internal metadata of the executable itself.

## Advanced Usage

**PRO TIP:** Watch for changes in the VolumeID, as these can be indicative of applications being run from external devices. In the example below, the VolumeID is different for each executable run, meaning that they were all run from different volumes even though two entries reference the E:\ drive.

| Volume ID | File ID Last-Write Timestamp | SHA1 | Full Path |
|---|---|---|---|
| abcd082d-3b8e-11e3-be8d-24fd52566ede | 10/23/2013 3:09 | f107ec56d650bf2cb00b186cbfbd202f66209ecf | E:\FTK Imager\FTK Imager.exe |
| afd25598-3b2c-11e3-be8c-24fd52566ede | 10/22/2013 21:42 | ca5fd519a43ff95d1ec0bbdf3533e9392109af74 | E:\TACTICAL Subject\f-response-tacsub.exe |
| dbcc2aeb-5826-41c0-8011-f0153438122b | 10/13/2013 9:42 | 9fef303bedf8430403915951564e0d9888f6f365 | C:\Windows\System32\notepad.exe |

**PRO TIP:** Looking for something specific in the Amcache? You can use the switches -b (blacklist) or -w (whitelist). Blacklisting will include only those Amcache entries that match the SHA-1 hashes specified in the file, while whitelisting will exclude those Amcache entries that match the SHA-1 hashes. In the example below, we've provided SHA-1 values in the Blacklist.txt, meaning that the output CSV will contain items that are only responsive to the SHA-1 values in the text file.

```
AmcacheParser.exe -f E:\Windows\AppCompat\Programs\Amcache.hve
-b G:\Blacklist.txt --csv G:\Amcache
```

# WxTCmd – Timeline Explorer

## Type of Artifact

The 1803 update of Windows 10 introduced the Timeline feature. This keeps a record of the last 30 days of applications and files opened by a given user. This can be seen by holding the Tab button and pressing the Windows button. The data for this are also synchronized from other computers where the user has logged in with their Microsoft account. The data for the Timeline is stored in a SQLite database.

## Basic Usage

WxTCmd takes a single ActivitiesCache.db file as input. If the input is coming from a mounted evidence item, it needs to be mounted as read-write/write-temporary. Output for this command is not output to the screen, so a CSV needs to be specified.

In the example command below, WxTCmd is being run against the ActivitiesCache.db file stored on an evidence file mounted as a disk (E:). Note that the subfolder named "a3936c317ac1474e" is not consistent. An equivalent, differently named folder will be present for other users.

```
WxTCmd.exe -f E:\Users\srogers\AppData\Local\
ConnectedDevicesPlatform\a393c317ac1474e\ActivitiesCache.db
```

## Key Data Returned

Processed Timeline data in a standardized CSV, XML, or JSON. There are several columns of potential interest in a forensic investigation. The "Executable" column provides the name and the path of the executable in use. For example, "Program Files x86\Adobe\Acrobat Reader DC\Reader\Acrord32.exe" would show that Acrobat Reader was opened. "Display Text" provides information regarding the content opened and the application used. For example, "Tax Documents.pdf (Acrobat Reader DC)" would indicate that the file "Tax Documents.pdf" was opened using Acrobat Reader. "Content Info" provides information relating to the location of the item that was opened. Following the same example as above, "C:\Users\lee _ w\Desktop\Tax Documents.pdf" would indicate the location of the file that was opened. There are also various dates and times recorded in the Timeline. "Start Time" indicates the first time, in the last 30 days, that this specific activity occurred.

## Advanced Usage

**PRO TIP:** Among the parsed data provided by WxTCmd is the column named "Content Info". As described above, this column contains the location and name of the opened file or resource. However, it also contains another valuable piece of information. In the example below, a file was opened from a "D:" drive. This ActivitiesCache.db file contains information for all computers synchronized to this Microsoft account, so several linked computers could have a "D:" drive. The example below provides the GUID (Global Unique Identifier) for the volume that stores that file. This means that the file can be tied back to a specific volume on a specific device.

```
D:\Files\Cat.jpg (file:Unmapped GUID: //D:/Files/Cat.
jpgVolumeId={A98818E7-5868-4C06-807E-0F24C9746829}&ObjectId=
{AE26BE95-ACAC-11E9-B3FB-60F6770E22E2})
```

# MFTECmd – MFT Explorer

## Type of Artifact

MFTECmd parses a number of different files from NTFS-formatted drives. At a high level, MFTECmd parses each of these internal NTFS System files. At a lower level, the application dives deep into NTFS and helps uncover much data of interest.

| File | Description | Contents |
|---|---|---|
| $MFT | Index of each file and folder on volume | File name timestamps, and other metadata |
| $Boot | Volume boor record | Volume serial nbr, volume signature, nbr of sectors |
| $SDS | File ownership | Contains a list of all the Security Descriptors on the volume |
| $J | USN Journal | Transaction log of all changes to a file (write, delete, rename, etc.) (file change journal) |
| $Logfile | Transaction Log File | Used by NTFS to maintain the integrity of the filesystem in the event of a crash (metadata change journal) |

## Basic Usage

MFTECmd takes a $MFT, $J, $SDS, $Logfile or $Boot as input. These input files can be in the form of an exported copy of the file(s) or by referencing them from within a mounted image. The example command below shows MFTECmd being run against a $MFT file that has been exported from an evidence file and the data being saved to a CSV file.

```
MFTECmd.exe -f 'G:\Exports\$MFT' --csv G:\MFT_Output
```

In the next example MFTECmd is run against a $MFT file stored on a mounted disk (E:) and outputting the data as a CSV file. In order to run this command, it is recommended to mount the evidence using Arsenal Image Mounter as write-temporary.

```
MFTECmd.exe -f 'E:\$MFT' --csv G:\MFT_Output
```

Note the command line syntax for referencing the alternate data streams $UsnJrnl and $Secure.

```
MFTECmd.exe -f 'E:\$Extend\$UsnJrnl:$MFT' --csv G:\USN_Output
```

```
MFTECmd.exe -f 'E:\$Secure:$SDS' --csv G:\SDS_Output
```

## Key Data Returned

The columns of most significance are highly dependent on the type of investigation and the reason for parsing the files in the first place. For example, the dates and times in the $MFT could provide an indication as to the copying of files from external devices. If the written/modification time precedes the creation time, there is a high degree of probability that the file was copied from another volume.

In the example below, the $MFT has been parsed to CSV and loaded into Timeline Explorer. In each row the Last Modified time precedes the Created time.

| Parent Path | File Name | Created0x10 | Last Modified0x10 |
|---|---|---|---|
| donald | | | |
| .\Users\Donald\Pictures | WP_20130805_001.jpg | 2013-08-12 01:11:19.3130517 | 2013-08-05 11:48:29.0000000 |
| .\Users\Donald\Pictures | WP_20130804_006.jpg | 2013-08-12 01:11:18.8755517 | 2013-08-05 00:55:58.0000000 |
| .\Users\Donald\Pictures | WP_20130804_005.jpg | 2013-08-12 01:11:18.4536393 | 2013-08-05 00:55:36.0000000 |
| .\Users\Donald\Pictures | WP_20130804_004.jpg | 2013-08-12 01:11:18.2192675 | 2013-08-04 16:41:50.0000000 |
| .\Users\Donald\Pictures | WP_20130804_003.jpg | 2013-08-12 01:11:17.9379831 | 2013-08-04 16:41:42.0000000 |
| .\Users\Donald\Pictures | WP_20130804_002.jpg | 2013-08-12 01:11:17.6410948 | 2013-08-04 13:53:58.0000000 |
| .\Users\Donald\Pictures | WP_20130804_001.jpg | 2013-08-12 01:11:17.2661017 | 2013-08-04 13:53:55.0000000 |
| .\Users\Donald\Pictures | WP_20130731_001.jpg | 2013-08-12 01:11:16.8754645 | 2013-07-31 14:38:46.0000000 |

This is a clear indication that these files were copied from another volume.

The processed $J data can be used to determine the date and time that specific actions were taken on a file. **These actions include (but are not limited to) creating a new file, making changes to a file, deleting a file, overwriting a file, and renaming a file. The $LogFile tracks changes to the information found in the MFT such as timestamps and other metadata.** In the example below follow the flow of activity the files recorded in $J. The first entry is for the creation of a file named $IT74KUZ, then data is added to the file before it is closed. Immediately afterwards, the file sdelete64.exe is renamed to $RT74KUZ before also being closed. This all happens within the same hundredth of a second as sdeleted64.exe being sent to the $Recycle.bin

| Update Timestamp | Name | Entry Number | Sequence Number | Update Reasons |
|---|---|---|---|---|
| 2018-01-08 01:18:19.5829828 | $IT74KUZ.exe | 1161 | 63 | FileCreate |
| 2018-01-08 01:18:19.5829828 | $IT74KUZ.exe | 1161 | 63 | DataExtend\|FileCreate |
| 2018-01-08 01:18:19.5860618 | $IT74KUZ.exe | 1161 | 63 | DataExtend\|FileCreate\|Close |
| 2018-01-08 01:18:19.5860618 | sdelete64.exe | 5899 | 13 | RenameOldName |
| 2018-01-08 01:18:19.5860618 | $RT74KUZ.exe | 5899 | 13 | RenameNewName |
| 2018-01-08 01:18:19.5985898 | $RT74KUZ.exe | 5899 | 13 | RenameNewName\|Close |

A few moments later, both files are deleted as the $Recycle.bin is emptied.

| | | | | |
|---|---|---|---|---|
| 2018-01-08 01:18:23.3099473 | $RT74KUZ.exe | 5899 | 13 | FileDelete\|Close |
| 2018-01-08 01:18:23.3111809 | $IT74KUZ.exe | 1161 | 63 | FileDelete\|Close |

The $SDS file allows us determine file ownership. For example, in the first screenshot below we see output from the parsed $MFT loaded into Timeline Explorer. Looking at the NTUSER.DAT entry we can see that the Security ID for this file is 8271.

| Security Id | Parent Path | File Name |
|---|---|---|
| | users\stack | ntuser.dat |
| 8271 | .\Users\Stack | NTUSER.DAT |

If we then go to the $SDS output and search for that same Security ID, we find that the NTUSER.DAT file is owned by the user with the Relative ID of 1001. If needed, we can take the SID and tied it to a username via the SAM Registry Hive.

| Id | Offset | Owner Sid | Group Sid |
|---|---|---|---|
| 8271 | | | |
| 8271 | 6343136 | S-1-5-21-3001495921-1769015868-3887507880-1001 | S-1-5-18 |

## Advanced Usage

**PRO TIP:** It is important to remember that NTFS stores two sets of dates and times in each $MFT entry. These are known as the Standard Information Attributes (SIA) and the FILENAME attributes. This means that each file and folder will have timestamps in both groups. These dates and times behave differently and can indicate when a file was truly created, not just what Windows reports. For example, in the table below we see a number of files stored under the Windows directory. The Created0x10 is the created date and time as stored in the SIA and Created0x30 relates to those stored in the FILENAME attributes.

As can be seen in the table, both dates and times are the same for the first two entries, but the third entry shows a FILENAME creation date that is much later than the creation date stored in the SIA. This may be an indication of manipulation of the SIA timestamp for the syncmon.exe file and would warrant further investigation.

| Created0x10 | Created0x30 | Path (combined from Parent Path and File Name) |
|---|---|---|
| 3/18/2019 09:17 | 3/18/2019 09:17 | C:\Windows\System32\cmd.exe |
| 3/18/2019 09:18 | 3/18/2019 09:18 | C:\Windows\System32\mountvol.exe |
| 3/18/2019 09:19 | 8/18/2019 01:12 | C:\Windows\System32\syncmon.exe |

**PRO TIP:** When an evidence file is mounted as a drive MTFECmd can also dive into the volume shadow copies and retrieve previous versions of the $MFT, the $J and $SDS files. This can be done by virtue of the switches --vss and --dedupe as demonstrated in the command below. The --vss switch tells MFTECmd to search in the volume shadow copies and the --dedupe switch stops MFTECmd from reporting duplicate entries found in the volume shadow copies.

```
MFTECmd.exe -f 'E:\$Extend\$UsnJrnl:$J' --csv G:\MFT_Output
--vss --dedupe
```

# JLECmd – JumpList Explorer Command Line Edition

## Type of Artifact

Jumplists store critical information about files and folders that have been interacted with using various GUI applications in Windows. Among other things, Jumplists contain information about the application used to open target files and folders and store metadata specific to those target items. Those metadata contain details such as file name and location, dates and times, etc. Parsing the Jumplist data can be difficult and time-consuming as they are stored in a format known as MS OLE Structured Storage files. JLECmd makes parsing this data simple and quick.

## Basic Usage

JLECmd takes either a single Jumplist file (relating to a specific application) or a directory of Jumplists as input. If parsing a single Jumplist, use the -f option. If parsing a directory of Jumplists, use the -d option. It is also suggested that the -q switch be used to avoid dumping all results to the screen (which can dramatically slow down JLECmd's execution time).

In the example command below, JLECmd is being run against a single Jumplist stored on an evidence file mounted as a disk (E:) Output is stored on the G: drive to the "Jumplists" folder. JLECmd will create an output file (CSV in this case) with the date and time in the file name.

```
JLECmd.exe -f E:\Users\Donald\AppData\Microsoft\Windows\Recent\
AutomaticDestinations\ff103e2cc310d0d.automaticDestinations-ms
--csv G:\Jumplists -q
```

In the example command below, JLECmd is being run against all automatic jumplist files stored for the user "Donald". Output is stored in the same folder as before. JLECmd will create an output file (CSV in this case) with the date and time in the file name.

```
JLECmd.exe -d E:\Users\Donald\AppData\Microsoft\Windows\Recent\
AutomaticDestinations --csv G:\Jumplists -q
```

## Key Data Returned

The JLECmd output contains two important categories of data, evidence of execution and evidence of file knowledge. The table below shows some of the more significant columns to include in your review.

| Column Name | Forensic Value |
|---|---|
| AppIdDescription | Human readable name for AppID |
| DestListVersion | Used with MRU to detemine most recentely opened file in the Jump List |
| MRU | Used with DestListVersion to detemine most recentely opened file in the Jump List |
| Path | Location and name of file opened |
| TargetCreated | Creation Timestamp of file referenced in JL |
| TargetModified | Modification Timestamp of file referenced in JL |

## Advanced Usage

**PRO TIP:** Watch for changes in the "DriveType", "VolumeSerialNumber" and "VolumeLabel" columns as the data in these columns can indicate whether files have been opened from external devices. In the example below, the change in these columns shows that a file was opened from the USB device named "FILES".

Additionally, the local path may show the same drive letter for multiple removable devices (e.g., F:\) but you should also review the volume serial number and the volume label to determine if the drive letter is associated with the same or different devices.

| Target Modified | Drive Type | Volume Serial Number | Volume Label | Local Path |
|---|---|---|---|---|
| 9/1/2018 16:53 | Fixed storage media (Hard drive) | 7E58AAB0 | Windows10_OS | C:\Users\srogers\Documents\NETFLIX SEC Filings\SEC-NFLX-1193125-12-53009.pdf |
| 9/27/2018 17:42 | Fixed storage media (Hard drive) | 7E58AAB0 | Windows10_OS | C:\Users\srogers\Documents\Netflix 3Q13 Conference Call Announcement 09 30 13.pdf |
| 9/3/2018 14:13 | Removable storage media (Floppy, USB) | B0A9FE90 | FILES | F:\Forms\fy08-form-10k.pdf |
| 9/1/2018 16:43 | Fixed storage media (Hard drive) | 7E58AAB0 | Windows10_OS | C:\Users\srogers\Documents\NETFLIX SEC Filings\SEC-NFLX-1065280-13-8.pdf |

A mapping of app_ids to app name can be found at **https://for500.com/appid.**

# RECmd – Registry Explorer Command Line Edition

## Type of Artifact

This command line tool is used to access, search and recover, and export any data found in the WIndows Registry. To grasp why this tool is so powerful, just think about searching and exporting registry in a consistent output format. It's no big deal to do this with other tools until you have to do exactly the same thing across tens, hundreds, or thousands of machines.

## Basic Usage

Search NTUSER.dat for the key name that contains "Dropbox"

```
RECmd.exe -f "C:\Temp\NTUSER.dat" --sk Dropbox
```

Search UsrClass.dat for the key value that contains "Dropbox"

```
RECmd.exe -f "C:\Temp\UsrClass.dat" --sd Dropbox
```

Search the directory registry_files for the key value that contains "Dropbox". The last write time is >= Startdate, and the value name contains either "AppName" or "DisplayName", so don't recover deleted keys and don't process log files.

```
RECmd.exe --d "C:\Temp\registry_files" --sk "Dropbox" --StartDate
"11/13/2014 15:35:01" --RegEx --sv "(App|Display)Name" --recover
false --nl
```

RECmd will replay and apply all registry hive logs automatically. Use --nl to suppress this.

## Search

- StartDate Start date: last write timestamps (UTC)
- EndDate End date: last write timestamps (UTC)
- MinSize Find values with data size >= MinSize (specified in bytes)
- sk Search for <string> in key names
- sv Search for <string> in value names
- sd Search for <string> in value record's value data
- ss Search for <string> in value record's value slack
- Regular expressions must of course be valid .net regular expressions
- If either the key or value has spaces in them, enclose in quotes
- To get default values, use a value name of "(default)"
- "--sX" are search options; they use the "contains" logic
- -sd will convert the compare values to ASCII and Unicode before doing comparison unless the"--l" literal switch is used

In the example command below, we are looking for large registry key (1MB and base64 encoded) that often contain malware. Deleted keys are also retrieved and parsed.

```
RECmd.exe -d "C:\Temp\registry_files" --minsize 1M --Base64
--recover true
```

To search for binary data in value data, simply string together the hex characters you want to find, separated by dashes (04-00-EF-BE, for example).

```
RECmd.exe -hive "C:\Temp\registry_files" --sd"
```

## Batch Mode

By default, batch mode utilizes the same plugins as found in Registry Explorer and works the same way. When used by RECmd, the data from the plugin will be normalized into a standard format for CSV output. When a plugin is used to process a key or key/value, the data generated by the plugin are also saved out to a CSV. In this way, it is very similar to exporting the data from Registry Explorer (albeit to Excel vs. CSV).

**Batch File**

***Header***

- Description: A general description of what this batch file is going to find
- Author: Name of this batch file (can be more, too, like contact information)
- Version: A version number that should be incremented as changes happen
- Id: A unique (across all other batch files) GUID (Global Unique Identifier) that identifies this batch file

***Keys collection – Each entry consists of:***

- Description: A user-friendly description of what this key will find. Can be anything from the key name to a friendlier description of what it means, etc.
- HiveType: The type of hive this entry corresponds to. Valid choices are NTUSER, SAM, SECURITY, SOFTWARE, SYSTEM, USRCLASS, COMPONENTS, BCD, DRIVERS, AMCACHE, SYSCACHE
- KeyPath: The path to the key to look for
- ValueName: OPTIONAL value that, when present, is looked for under KeyPath
- Recursive: Whether or not to process KeyPath recursively
- Comment: Like Description in that you can add various things here that end up in the CSV

HiveType determines which kind of hive the entry corresponds to. This saves time in that RECmd won't search a SOFTWARE hive for keys that won't ever exist (because they are NTUSER-specific, for example).

***Batch File Example***

Detailed, fully functional example batch files can be found in the **ZimmermanTools\RegistryExplorer\BatchExamples** folder.

Wildcards are supported in the KeyPath within the batch file. Example:

```
SOFTWARE\Microsoft\Office\*\*\User MRU\*
```

To use batch mode, supply the file to the --bn switch, along with --csv to tell RECmd where to save results:

- Export UserAssist data via RECmd batch file that uses a Registry Explorer plugin

```
RECmd.exe --bn .\BatchExamples\BatchExampleUserAssist.reb -f
C:\Temp\NTUSER_dblake.DAT --nl --csv C:\Temp
```

- Export Registry many of the Registry Explorer Plugin CSVs using a batch file

```
RECmd.exe --bn .\BatchExamples\RECmd_Batch_MC.reb -d G:\blake\
Registry\E --nl --csv g:\blake\recmd_out
```

**PRO TIP:** Be as specific as possible about the directory to process as it can have a significant impact on performance. These two commands generate the same results but the second one runs much faster.

This is much slower because the RECmd has to process the entire drive.

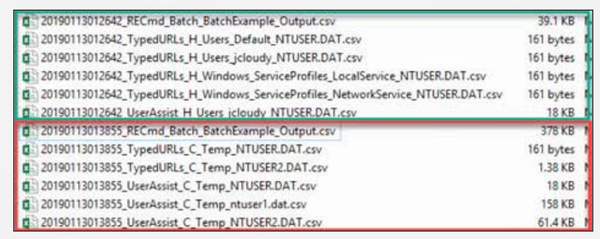
```
RECmd.exe --bn "C:\Forensic Program Files\ZimmermanTools\
RegistryExplorer\BatchExamples\UserActivity.reb" -d G:\blake\
Registry\E --nl --csv g:\blake\registry\recmd_out
```

This is much faster because RECmd is only processing a single user directory

```
RECmd.exe --bn "C:\Forensic Program Files\ZimmermanTools\
RegistryExplorer\BatchExamples\UserActivity.reb" -d G:\blake\
Registry\E\Users\Donald --nl --csv g:\blake\registry\recmd_out
```

**PRO TIP:** A RECmd batch file can contain instructions for processing different Hives & Keys. Using the -f option allows you to target a specific hive instead, if desired, all hives mentioned in the batch file.

When RECmd runs in batch mode, several files will get generated in the --csv directory (see the example to the left).




A mapping of app_ids to app name can be found at **https://for500.com/appid.**

# LECmd – LNK File Explorer

## Type of Artifact

Shortcut files (*.lnk) are shell items and, as such, not entirely human-readable. Lnk files are most frequently created when a user opens a non-executable file by double-clicking. These shortcut files are stored under the user profile that opened the file and contain information relating to the opened target file. This includes information such as the target file dates and times (at the time when the file was opened), file name and path, the drive type, volume serial number, volume label and more. LECmd takes this data and presents it in a human-readable format.

## Basic Usage

LECmd takes, as input, either a single lnk file or a folder containing several such files.

In the example command below, LECmd is being run against a single lnk file stored on an evidence file mounted as a disk (E:). When running this command the output is shown in the window running the command (command line window or PowerShell).

```
LECmd.exe -f E:\Users\srogers\AppData\Microsoft\Windows\Recent\
Peggy.jpg.lnk
```

In the next example, LECmd is being run against a folder of lnk files stored on the same mounted evidence file as above. This time, the output is stored in a CSV stored in G:\LnkFiles.

```
LECmd.exe -f E:\Users\srogers\AppData\Microsoft\Windows\Recent
--csv G:\LnkFiles -q
```

## Key Data Returned

| Column Name | Forensic Value |
|---|---|
| AppIdDescription | Human readable name for AppID |
| DestListVersion | Used with MRU to detemine most recentely opened file in the Jump List |
| MRU | Used with DestListVersion to detemine most recentely opened file in the Jump List |
| Path | Multiple Path Columns: Location and name of source and target files |
| SourceCreate | Creation Timestamp of the LNK itself |
| SourceModified | Modification Timestamp of the LNK itself |
| TargetCreated | Creation Timestamp of target file the LNK points to |
| TargetModified | Modification Timestamp of target file the LNK points to |
| DriveType | Network, fixed loal, ior Removable |
| VolumeSerialNumber | MFT Entry Number |
| MFT Nbr & Seq nbr | MFT - Seg nbr - If present then Voluome is NTFS |

## Advanced Usage

**PRO TIP:** Taking the data from key columns not only tells a forensic investigator when the file was opened, but may also provide details about the number of times a user accessed a file with that name. In the table below, the first row of results indicates that the file was only opened once, as SourceCreated and SourceModified contain the same time. The second instance indicates that the file has been opened at least twice, as the SourceCreated occurred around seven hours before the SourceModified. We also see that the Target dates are identical, suggesting that the file has not been changed since it was created. The last row indicates that the file was only opened once, since the Source entries are identical, However, the TargetModified precedes the TargetCreated, indicating that the file has been copied to the F: drive from another location.

| Source Created | Source Modified | Target Created | Target Modified | Path (Combined from Local Path and Common Path) |
|---|---|---|---|---|
| 9/1/2018 16:53 | 9/1/2018 16:53 | 8/27/2018 09:24 | 9/6/2018 14:43 | C:\Users\Donald\Documents\NETFLIX SEC Filings\SEC-NFLX-1193125-12-53009.pdf |
| 9/27/2018 10:42 | 9/27/2018 17:37 | 9/27/2018 10:28 | 9/27/2018 10:28 | C:\Users\srogers\Documents\Netflix 3Q13 Conference Call Announcement 09 30 13.pdf |
| 9/3/2018 14:13 | 9/3/2018 14:13 | 9/3/2018 14:11 | 9/1/2018 18:19 | F:\Forms\fy08-form-10k.pdf |

**PRO TIP:** LNK facts to keep in mind:

- The target file name extension is not always provided in the LNK name.
- The LNK file points to the last file of that name. Meaning, if there were two files named exactly the same, the link files point to the last one opened.




This poster was created by Mark Hallman and Lee Whitfield with support from the SANS DFIR Faculty