"Evidence of" Example

Example of filtering for evidence of program execution. This filter identifies more events than using parsers. Long command lines can exceed the Windows command line limit and if so, will have to be run in Linux.

```
psort.py -z "UTC" -o l2tcsv -w
execution_of_execution.csv c-drive.plaso
"message contains 'Prefetch {' or
message contains 'AppCompatCache' or
message contains 'typed the following
cmd' or
message contains 'CMD typed' or
message contains 'Last run' or
message contains 'RunMRU' or
message contains 'MUICache' or
message contains 'UserAssist key' or
message contains 'Time of Launch' or
message contains 'Prefetch' or
message contains 'SHIMCACHE' or
message contains 'Scheduled' or
message contains '.pf' or
message contains 'was run' or
message contains 'UEME_' or
message contains '[PROCESS]'"

psort.py -z "UTC" -o l2tcsv -w
execution_test.csv c-drive.plaso
"parser is 'userassist' or parser is
'prefetch' or parser is 'amcache' or
parser is 'windows_run'"
```

Context Sensitive Fields

LNK files – drive_serial_number, driv_type, volume_label
Prefetch – executable, mpaaed_drives, mapped_files, volume_serial_serial_numbers
EVTX – event_identifier, source_name, message_string

These are just a few examples, there are many more. These context sensitive fields where found by reviewing the Plaso formatters on the Plaso GitHub page.

Windows Data_Types

```
registry:key_value
windows:distributed_link_tracking:creat
ion
windows:evtx:record
windows:lnk:link
windows:metadata:deleted_item
windows:prefetch:execution
windows:registry:amcache
windows:registry:amcache:programs
windows:registry:appcompatcache
windows:registry:installation
windows:registry:key_value
windows:registry:list
windows:registry:network
windows:registry:office_mru
windows:registry:sam_users
windows:registry:service
windows:registry:shutdown
windows:registry:userassist
windows:shell_item:file_entry
windows:srum:application_usage
windows:srum:network_connectivity
windows:srum:network_usage
windows:tasks:job
windows:volume:creation
```

Data_types can provide a much finer level of granularity than parsers. There are many other data_types. Take a look here. https://github.com/mark-hallman/plaso_filters

Data_Type Filter Examples

```
$ psort.py -o l2tcsv -w userassist.csv
c-drive.plaso "data_type is
'windows:registry:userassist'"

$ psort.exe -z "UTC" -o l2tcsv -w
files_on_usb.csv c-drive.plaso
"data_type is 'windows:lnk:link' and
drive_type == 2"

$ psort.exe -z "UTC" -o l2tcsv -w
chrome.csv c-drive.plaso "data_type
contains "chrome"
```

** "drive_type" is an example of a "context "sensitive field, meaning it is only available for certain types of events. In this case, LNK file events. Drive_type == 2 is for removable drives. More examples at:

SANS DFIR
DIGITAL FORENSICS & INCIDENT RESPONSE

Plaso Filtering Cheat Sheet 1.03

Timelines are crucial to DFIR analyst's efforts to paint a picture of what happened on a device or in an incident. Plaso is a widely adopted tool for creating timelines. If constraints are not focus results Plaso can generate overwhelming amounts

How To Use This Sheet

This document is aimed to be a reference on the filtering options available with each of the Plaso tools. Although there is some overlap in filtering options across the various tools, there are also filtering options that are unique to a specific tool. There are also filtering options that are not widely documented and are shown here. There are some lists of items, such as data_types, that are not shown in their entirety. Complete Lists can be found at:

https://github.com/mark-hallman/plaso_filters

image_export

Files can be extracted by filter file, extension, date filter, signature. The Filter File is the same format as the file used for log2timeline.

```
$ image_export -f filter_windows.txt
--no_vss  -w export_folder_name c-
drive.e01
```

Timestamp types: atime, ctime, crtime, bkup

```
$ image_export.py -vss_stores all
-x "doc,docx,xls,xlsx,ppt,pptx,pdf"
--date_filter "crtime, 2013-10-21,
2013-10-23" -w c-drive_docs_export c-
drive.e01
```

log2timeline

Log2timeline Filtering Options: 1. File filters and 2. Parsers. These options can significantly decrease the number of events returned and time to execute. Eg. 2.5 hours down to 2.5 minutes.

Example filter files can be found at:
https://github.com/mark-hallman/plaso_filters

Get help and list all the parsers with:

```
$ log2timeline.py --info
```

Use filter file and process no VSS's:

```
$ log2timeline.py -f
filter_windows.txt
--no_vss c-drive.plaso c-drive.E01
```

Use filter file, process All VSS's (and live) and use a list of parsers

```
$ log2timeline.py -f
filter_windows.txt –parsers
"amcache,prefetch,userassist"
--vss_stores all c-drive.plaso c-
drive.E01
```

Source does not have to be an image

```
$ log2timeline.py triage.plaso
/mnt/windows_mount
```

psort

Output Formats

```
$ psort.py -o list –
```
Shows all available formats

Commonly used output formats

l2tcsv – 17 field legacy log2timeline fixed format

```
date,time,timezone,MACB,source,sourcetyp
e,type,user,host,short,desc,version,file
name,inode,notes,format,extra
```

dynamic – default output 9 fields. Fields can be added or removed from this format.

```
datetime,
timestamp_desc, source,
source_long,message, parser,
display_name,tag
```

dynamic output examples using –-fields & --additional_fields

```
$ psort.py -z "UTC" -o dynamic —
additional_fields
"data_type,strings,event_type" -w
add_fields.csv
c-drive.plaso

$ psort.py -z "UTC" -o dynamic --fields
"datetime,macb,data_type,drive_serial_nu
mber,drive_type" -w winlnk.csv c-
drive.plaso "data_type is
'windows:lnk:link'"
```

Filter on fields that are not in output format

```
$ psort.py -z "UTC" -o l2tcsv -w
winlnk.csv c-drive.plaso "data_type is
'windows:lnk:link' and drive_type == 2"
```

Start with date as a filter. Best for larger ranges.

```
psort.py -z "UTC" -w date_filtered.csv
c-drive.plaso "date > '2018-10-11
00:00:00' AND date < '2018-10-22
023:59:59'"
```

Time Slice – Best for smaller, targeted, ranges.

```
psort.py -z "UTC" --slice '2018-10-22
010:59:59' –slice_size 1 -w sliced.csv
c-drive.plaso
```

Slicer – Event context- Nbr of events surrounding each filtered event

```
psort.py -z "UTC" --slice_size 20 --
slicer -w slicer.csv c-drive.plaso
```

Filtering Tips

- Parsers and file filters with log2timeline are a good practice most of the time.
- "contains" == case insensitive "is" == case sensitive
- No parsers == default to "win7"
- data_types are all lower case.
- All commands are shown with the .py as run from Ubuntu. Windows version has a .exe extension
- Image_export – easy way to get files out of VSS's
- Plaso runs very well in Windows. No VM, simple to install and you have easy access to your other Windows tools.
- "date" used in filters is the date field in the default (dymanic) output
- Multiple psort output files (csv) can be concatenated if you have filters that can't be expressed in a single statement.

Event Tagging

Tagging populates the "tag" field in the Plaso DB based upon rules define in the tag file. That tag value can then be used to filter. Tags are assigned to events based upon rules defined in the tagging file. An event can be responsive more than one tag rule or to no rule at all. Events that are responsive to more than one expression will have a tag value similar to (tag1,tag2,tag5). The tag field can included in your output when using the Dynamic output format (-o dynamic)

The message, also referred to as long_desc, can't be used in a tagging file expression.

Run the tagging process with tag_file

```
psort.exe -o null --analysis tagging
--tagging-file tag_windows.txt -w c-
drive.plaso
```

Use the tags that were populated in the step above to filter

```
psort.exe -o l2tcsv -w
```