TZWorks® EVTX Fragment eXtension (*evtfx*) Parser Users Guide

Abstract

evtfx is a standalone, command-line tool that can extract and parse EVTX type records from fragmented or corrupted Eventlog files. The tool can report the results in a CSV type format or SQLite database. It has compiled version for Windows, Linux and OS-X.

www.tzworks.com

Contact Info: info@tzworks.com

Document applies to v0.12 of ***evtfx***

Updated: Aug 24, 2022

Table of Contents

TZWorks® EVTX Fragment eXtension (evtfx) Parser Users Guide

Webpage: http://www.tzworks.com/prototype_page.php?proto_id=53
Contact Information: info@tzworks.com

1 Introduction

The Windows operating system uses binary XML notation to record various events that occur during the normal course of system usage. In forensics, use of the data recorded in the Window's event log is extremely useful in determining the changes occurring in a machine over time periods of interest. There are many utilities that allow one to extract records from these same logs and package them by time or event to highlight activities that occurred. Occasionally, however, when a parser encounters a corrupted log file, parsing out records can be problematic at best. Therefore, the objective for ***evtfx*** was to focus on this area; parsing event logs that were either: (a) corrupted internally either intentionally or accidently by the system (during a crash) or (b) were partially recovered usually from a file carving operation, but still contained gaps in the data.

In order to design a tool that can parse EVTX type records from corrupted or partial fragments of a log, one needs to adjust the way a normal EVTX type parser works. As background, EVTX type logs, as part of their internal design, attempt to minimize the space usage by incorporating a position dependent record structure. Specifically, one record can rely on another record's definitions of field names or data values. This allows a reduction of space and acts as a compression technique when storing many similar events; many of the data labels are the same and will reference a main record containing a shared template definition. While this is very useful under normal circumstances, unfortunately, when the records become corrupted or deleted, this position dependence can cause undefined behavior for any parser relying on traversing the position related pointers. Case in point is the evtwalk tool from *TZWorks*; it does an excellent job at parsing uncorrupted records, however, does a *best guess* type parsing for those records that are deleted (using the *-inc_slack* option, which means to include the slack space during the parsing operation). Furthermore, if for whatever reason, the parser cannot find the template definition, it won't parse records that rely on the missing definition. By extension, any records that are outside the log file structures usually get missed as well.

The purpose of ***evtfx*** is to address the shortfalls of EVTX parsers when it comes to corrupted or partial event logs. As part of its architecture, it is designed to be somewhat independent over the state of the previous records. This, in turn, allows adds robustness for handling missing/corrupted records.

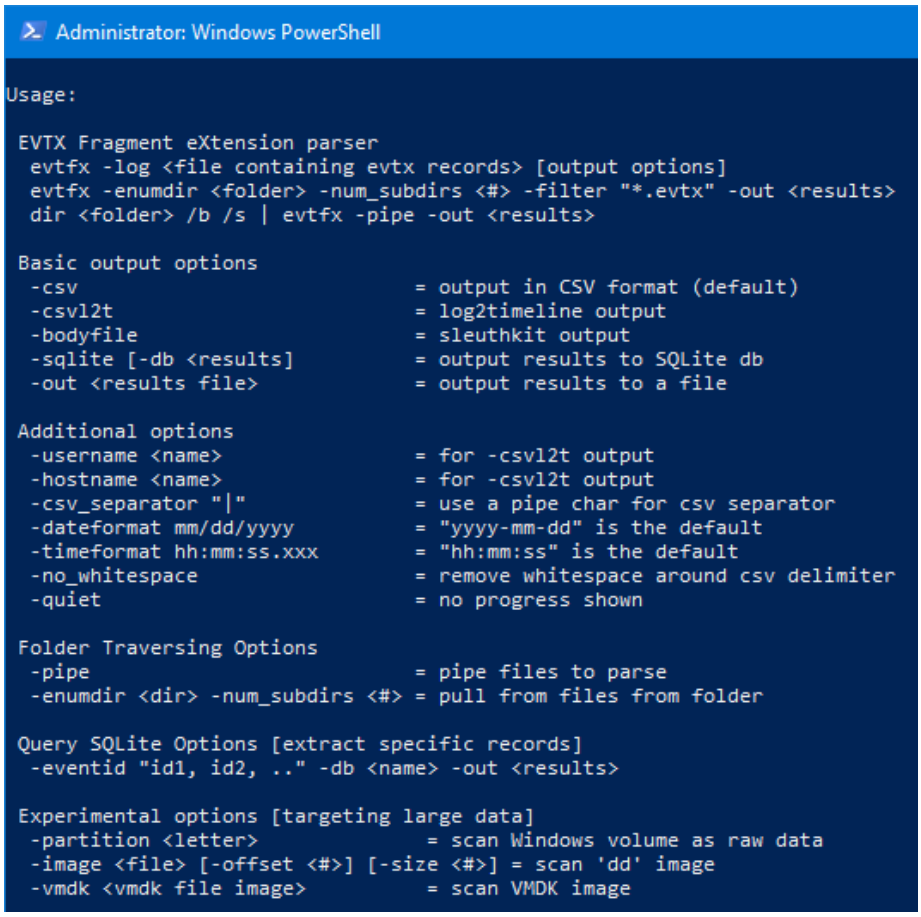
The downside to the ***evtfx*** tool, is it is slower in actual parsing time when compared to ***evtwalk.*** In some cases, the ***evtfx*** tool misses some of the content of the data. In general, the accuracy of the results

of ***evtfx*** comes close to that ***evtwalk***. Therefore, if the event log is normal and intact, the ***evtwalk*** tool should be the tool of choice. If, however, one wishes to extract corrupted or deleted records from an event log, the ***evtfx*** tool is good choice.

Taken to the extreme, the ***evtfx*** tool can be used to pull out EVTX type records from any blob of data, assuming the data is uncompressed and unencrypted. The latter condition is very important, since Windows may incorporate NTFS file compression for the event log in question, in which case, the ***evtfx*** tool may fail to parse the compressed data. While ***evtfx*** makes some attempts to try to perform NTFS file decompression, the results are much more reliable if the raw cluster data is uncompressed.

2 How to Use *evtfx*

One can display the menu options by typing in the executable's name without parameters. Below is the menu with the various options.



To process a file with EVTX type event log records, use the ***-log <file>*** notation. If sending the output to a file, use the ***-out <results>*** option along with the type of desired delimited format (***-csv, -csvl2t or -bodyfile***). Alternatively, one can send the data to a SQLite database; to do this, use the notation ***-sqlite -db <database to store results>***.

The other options shown above (under *Additional options*) are the standard ones used in many of the *TZWorks* tools and contain the same behavior as the other tools. Details about these options are contained in the section on "*Available Options*".

The delimited output formats (with the exception of ***-csvl2t***) generate one record per log entry. The delimiter for all the output file formats can be specified by the user. Some of the more common ones include either a *comma character*, *pipe character* or a *tab* character. If one tells ***evtfx*** to send the results to a SQLite database, then the tool will either create a new database if none exists, or, if one exists with the same name, the results will be appended to the existing one. Later on, one can query the database and extract desired records either by using another tool, such as the "DB Browser for SQLite" or using

standard SQL commands. There is a rudimentary query option built into ***evtfx*** to extract specific event identifiers using the ***-eventid <#1, #2, ...>*** option which is discussed in more detail in the section of *Extracting Specific Event IDs*.

2.1 Processing Multiple Event logs (or log fragments)

If desiring to process many log files in one pass, one can put the artifact logs/fragments into separate a subdirectory and use the ***-pipe*** option like so:

```
>dir c:\dump\eventlogs /b /s /a | evtfx64 -pipe -out results.csv
```

Alternatively, if not wishing to use the piping option, one can use the ***-enumdir*** option along with the sub options ***-num_subdirs*** and ***-filter***. This allows one to target a certain level of subdirectories and only files with the desired extension.

```
>evtfx64 -enumdir c:\dump\logfrags -num_subdirs 2 -filter "*.bin"  -out results.csv
```

The above command will process all the logs and/or fragments contained in the *c:\dump\logfrags* folder and subfolders down to two subdirectories with the filename extension ".bin". The output will be stored in the *result.csv* file.

2.2 Scanning large datasets

Internally, ***evtfx*** will try to scan log files by looking for common EVTX log magic signatures. These magic signatures can either be the start of a log file (*ElfFile*) or a chunk at one of the internal sections (*ElfChnk*). This approach allows the ***evtfx*** to scan the data more quickly than the alternative of looking for a *record signature* from unrelated data areas. The tool will automatically shift to *record signature* scans for each internal *ElfChnk* signature found, or if the fragment being analyzed is less than the size of the *ElfChnk*.

The tool has had some limited testing against partitions and memory dumps. These options are considered experimental, but can be invoked via the ***-partition <letter>***, or just ***-image <memory dump>***. In cases where the data analyzed is not compressed or encrypted, ***evtfx*** does a relatively good job at extracting complete event log records. In some cases, when the ***evtfx*** cannot recognize the template used in the record, it will try to translate the fields either using the slot/index notation or pattern-match the sequence into another template with the same pattern. While the translation is usually correct, occasionally, the template names chosen for the fields are incorrect. This happens infrequently; as more testing is done the algorithm can be improved to try to eliminate the false-positives. For those cases, where ***evtfx*** cannot determine the template translation at all, it will resort to using field names

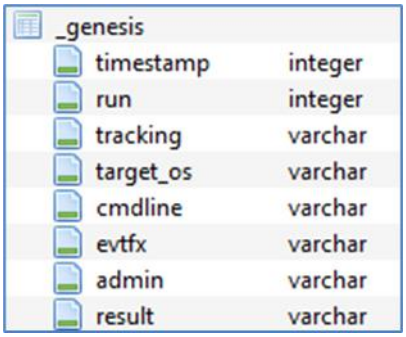
like "*slot_00, slot_01, …, slot_xx*" and leave it up to the analyst to determine the translation of the data fields.

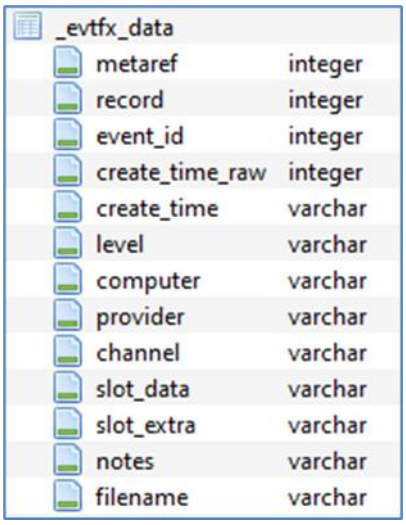
The other point to make is that ***evtfx***, when parsing data from raw clusters from a partition or memory dump, does not try to reconstruct the cluster run for the event log file. Instead, it just traverses one cluster, sequentially, when it looks for EVTX records. It is an enhancement that can be added later, if required. The primary intent of these two options (***-partition*** and ***-image***) was a way to stress the tool so it could encounter differing levels of corruption in the records during the parsing operation. The idea was to force ***evtfx*** to encounter as many boundary conditions as possible so they could be addressed during testing prior to release. These options were left in so the analyst could play with them as well (and hence the reason they are labelled *experimental*).

Finally, when processing large datasets, whether it be from many event log files in one session or targeting an entire volume that is large, the results produced can potentially create a very large results file (or database, if using the SQLite option). Keep this in mind with processing many files at once, since handling a very large CSV may not be something that is desirable. For this reason, the SQLite option was added, to make it more extensible for the user to query the final results relatively quickly.

3 SQLite Results

If one chooses to output the parsed data into a SQLite database, the two tables of interest are the *_genesis* and the *_evtfx_data* tables. The first table (*_genesis*) stores the command line parameters used when running ***evtfx*** along with some other metadata about the system when generating the results in the *_evfx_data* table. The second table (*_evfx_data*) contains a record for each event that was parsed. The fields for each of these tables are shown below:

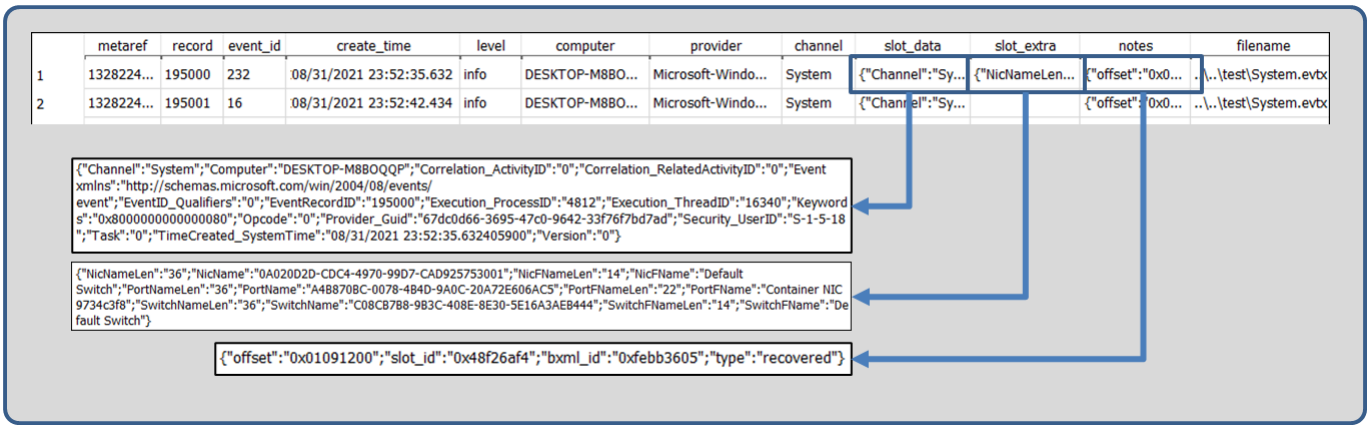




For each event record that is parsed, there are some fixed value fields and there are some variable fields. The fixed value fields are ones that are common across many events (eg. *record, event_id, create_time*, etc.), while the variable ones are unique to the specific event identifier/type. Unique data occurs in the slot values of the record, where each value can have a context specific name and is dictated by the template definition referenced by the event. Since JSON is just a set of key/value pairs, it offers the flexibility to capture all the event log data independent to place into these variable fields. The s*lot_data, slot_extra*, and *notes* are the variable type fields and have key/value pairing of data.

Below is an example of parsing a *System.evtx* log and how the records get translated into the SQLite database schema. To start the example, a normal view of the data in *XML format* is shown for reference purposes. Then, a screenshot of how this same data is translated into the *_evtfx_data* table as a record.

```
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
<System>
<Provider Name="Microsoft-Windows-Hyper-V-VmSwitch" Guid="67dc0d66-3695-47c0-9642-33f76f7bd7ad"/>
<EventID Qualifiers="0">232</EventID>
<Version>0</Version>
<Level>4</Level>
<Task>0</Task>
<Opcode>0</Opcode>
<Keywords>0x8000000000000080</Keywords>
<TimeCreated SystemTime="08/31/2021 23:52:35.632405900"/>
<EventRecordID>195000</EventRecordID>
<Correlation ActivityID="0" RelatedActivityID="0"/>
<Execution ProcessID="4812" ThreadID="16340"/>
<Channel>System</Channel>
<Computer>"DESKTOP-M8BOQQP"</Computer>
<Security UserID="S-1-5-18"/></System></Event>
<EventData>
<Data Name="NicNameLen">36</Data>
<Data Name="NicName">0A020D2D-CDC4-4970-99D7-CAD925753001</Data>
<Data Name="NicFNameLen">14</Data>
<Data Name="NicFName">Default Switch</Data>
<Data Name="PortNameLen">36</Data>
<Data Name="PortName">A4B870BC-0078-4B4D-9A0C-20A72E606AC5</Data>
<Data Name="PortFNameLen">22</Data>
<Data Name="PortFName">Container NIC 9734c3f8</Data>
<Data Name="SwitchNameLen">36</Data>
<Data Name="SwitchName">C08CB7B8-9B3C-408E-8E30-5E16A3AEB444</Data>
<Data Name="SwitchFNameLen">14</Data>
<Data Name="SwitchFName">Default Switch</Data></EventData>
```



The *slot_data* field contains the normal (*non-binary XML*) slot data found in the record. The *slot_extra* field contains the *binary XML* stream that may or may not be embedded into one of the normal slots, and it contains its own set of slot data (along with its own template reference). These are broken out as two separate fields primarily for debugging purposes. The last variable field is for general *notes*, which is used to assist in validation of the parsed record. It contains the offset the record found along with the template identifiers for the normal slot data and the *binary XML* slot data (if it exists).

The other table that is of use is the *_genesis* table. It describes the metadata associated with the specific running of the ***evtfx*** tool. Below is an example of the output.

	timestamp	run	tracking	target_os	cmdline	evtfx	admin	result
1	13282251...	1	0x1d6...	Windows;DESKTOP-M8BOQQP;10.0.-;10.0.1;64	evtfx64 -log ..\..\test\System.evtx -sqlite ..\..\test\system_test.db	ver: 0.01-64	n	0

If one runs ***evtfx*** multiple times, sending the output to the same SQLite database, the tool will append new records to the *_genesis* and *_evtfx_data* tables. The *_genesis* table will record each time the tool ran, and the *_evtfx_data* will store each parsed event log record. The *timestamp* field in *_genesis* uses the Windows *filetime* epoch. Likewise, the *metaref* field in *_evtfx_data* also uses the Windows *filetime* epoch with some additional ticks to avoid collisions between entries (since sending the output to the database is done in bulk and results in each entry being submitted faster than the resolution of the timestamp). Using these two fields from their respective tables, one can synchronize on time, to separate which records were parsed for each instance that was run by the tool, if that was of interest.

3.1 Extracting Specific Event IDs

If a SQLite database was created to store ***evtfx*** results, then one can go back and either query the database using a SQL statement or using the build-in ***-eventid*** command. A typical SQL statement to extract a specific event identifier could be:

```
>SELECT * FROM _evtfx_data WHERE event_id IN (4614, 4723, 4724, 4738)
```

The example above assumes one is in a SQL shell and loaded the database that was generated by ***evtfx***. The following entries in the above command mean:

- *_evtfx_data* = table name
- *event_id* = field name within the table to filter on
- 4614, 4723, 4724, 4738 = event id's that relate to records where the password changes in the security log. These are the ones we would like to extract.

Alternatively, one could use the ***-eventid*** command built into ***evtfx***, and accomplish a similar result:

```
>evtfx64 -eventid "4614, 4723, 4724, 4738" -db security_frag.bin -out results_pw.csv
```

The advantage of this last option is one does not need a SQL shell, but can run the extraction directly using the ***evtfx*** tool and output the results into an output file with CSV formatting.

To see other categories of event ID combinations associated with system changes, see the section on "Event Category Reports" in the ***evtwalk*** user's guide.

4 EVTX Record Templates

Many EVTX records, with some exceptions, make common use of template definitions to specify how to interpret the field labels associated with the value data. During the parsing process a normal EVTX parser would look for the template definitions within each *ElfChnk* data section. This works fine if all the data is in order and the *position of the records relative to the start of the template definition are preserved*. This type of parsing relies on *position dependent* translation of the data field. However, when considering the case where the data may not be contiguous, or if the template definition is corrupted, then the problem becomes more difficult and the *position dependent* parsing fails.

The parsing engine in ***evtfx*** uses a couple of techniques to get around the *position dependent* parsing. Since each record contains a pattern that is associated with a template definition, in combination with a template identifier that is embedded into each record, the tool then can simply do a lookup on these parameters to derive which field names are associated with the values when doing the translation. However, this requires one to store the template definitions within the tool. Unfortunately, there are literally thousands of template definitions, therefore, storing them all is not an acceptable option. Alternatively, what ***evtfx*** does is store some of the more common template definitions to handle the general cases. To handle all the rest of the template definitions, ***evtfx*** dynamically builds a template database on the fly when it encounters any record that contains a template definition. It stores this in an internal, dynamically built database that resides in memory, and accesses it during the parsing/translation process. While testing is still being done, empirical testing suggests, that just by using this approach, one can achieve very accurate results.

In conclusion, ***evtfx*** has a use-case that allows it to fit in to the EVTX set of parsing tools. With that in mind, the tool is still doing something non-standard, as far as the parsing process. This means it can result in errors in the translation process and the results produced by the tool should be considered experimental. For those wanting a more reliable event log parser than what ***evtfx*** offers, then one should consider using ***evtwalk*** tool, and limit usage of the ***evtfx*** tool for those cases where the log file cannot be parsed by other log parsing tools.

5 CSV Field Names / Meaning

Below is a refence of all the CSV fields used and their meanings.

CSV Field	SQLite Field	Definition
	metaref	Timestamp (using filetime epoch) with tick to ensure no aliasing of time
Record#	record	Record number in the event log
EventID	event_id	Event ID for the record
-	create_time_raw	Filetime (in epoch format) of the event
Create Time [UTC]	create_time	Date/Time in UTC format of the event
Level	level	Severity level of the event
Computer	computer	Computer name where the event occurred

ProviderName	provider	Event type provider
	channel	
SlotData	slot_data	Pairs of key/value name data associated with the event
SlotExtraData	slot_extra	Translation addition binary XML data embedded as a slot as pairs of key/value name data.
Notes	notes	Addition metadata, offset of the record and template identifiers
Filename	filename	Filename of the event log or fragment of logged parsed

6 Limitations

This version of the tool has a number of limitations. They are listed below.

- It is still prototype in nature being that this is the first version released. It still needs to be tested against various types of files, corrupted files, etc. to ensure the tool can perform consistently.
- Only parses records in event logs that are not compressed or encrypted
- Only works on EVTX type logs. Does not currently support the older WinXP logs.
- In order to translate an event log record, the template definition associated with that record needs to be located. If the template cannot be found, it needs to be derived. Sometimes this can cause errors in the translation of the output. For those cases, where ***evtfx*** cannot come up with anything, it will resort to using labels like (*slot_00, slot_01, …, slot_xx*).
- When parsing fragment files with corrupted records sometimes the parsing engine will encounter a boundary condition in the code logic and come to an abrupt stop. As ***evtfx*** matures, these boundary conditions are eliminated one by one.

7 Available Options

Option	Description
-log	Identify which event log(s) to operate on. The syntax is: ***-log <eventlog to analyze>***. To operate on more than one event log at a time, use: ***-log "<eventlog1> | <eventlog2> | ..."***
-db	Specifies which SQLite database to create or to act on. The format is: ***-db <name>***. During creation, one uses the ***-sqlite*** command in conjunction with the ***-db <name>***. During query, one uses the ***-eventid <ids>*** in conjunction with the ***-db <name>***
-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. If using the body-file in conjunction with the mactime.pl utility, one needs to set the environment variable TZ=UTC.
-sqlite	Outputs the results into a SQLite database. Requires the ***-db <name>*** specifier. The format is: ***-sqlite -db <name>***
-username	Option is used to populate the output records with a specified username. This only applies to the ***-csvl2t*** option. The format is: ***-username <name to use>***.
-hostname	Option is used to populate the output records with a specified hostname. This only applies to the ***-csvl2t*** option. The format is: ***-hostname <name to use>***.
-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 option. The syntax is *-filter <"*.ext | *partialname* | ...">*. The wildcard character '*' is restricted to either before the name or after the name.
-no_whitespace	Output the date using the specified format. Default behavior is *-dateformat "mm/dd/yyyy"*. This allows more flexibility for a desired format. For example, one can use this to show year first, via *"yyyy/mm/dd"* or day first, via *"dd/mm/yyyy"*, or only show 2 digit years, via the *"mm/dd/yy"*. The restriction with this option is the forward slash (/) symbol needs to separate month, day and year and the month is in digit (1-12) form versus abbreviated name form.
-csv_separator	Only applies to ***-csv*** and ***-csvl2t*** options. 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 restriction with this option is the colon (:) symbol needs to separate hours, minutes and seconds, a period (.) symbol needs to separate the seconds and fractional seconds; the repeating symbol 'x' is used to represent number of fractional seconds.
-quiet	Show no progress during the parsing operation.
-eventid	Extract the specified event IDs from the SQLite database (use ***-db <name>***). If more than one ID is specified, one needs to delimit each ID with a comma. The syntax is ***-eventid "id1, id2, ..."***.
-partition	Experimental. Extract EVTX records from the specified volume. The format is: ***-partition <volume letter>.***
-image	Experimental. Extract EVTX records from the specified file image. This could be a 'dd' image of a volume or a memory dump. As long as the EVTX logs are stored without any compression or encryption, the tool should be able to pull out and parse EVTX records. The format is: ***-image <file>.*** If the file is a 'dd' image of a disk one can add the offset/size within the image of the volume to analyze, using the sub options: ***-offset <#> -size <#>.***
-vmdk	Experimental. Extract EVTX records from the specified VMDK disk image. The format is: ***-vmdk <disk image>.*** One can add the offset/size within the image of the volume to analyze, using the sub options: ***-offset <#> -size <#>.***
-offset	Used to specify a starting offset to look at EVTX records. The format is: ***-offset <#>***
-size	Used to specify a size to look at EVTX records. The size is relative to the starting offset. The format is: ***-size <#>.***
-base10	Ensure number for sizes and addresses are displayed as base-10 format versus hexadecimal format. Default is hexadecimal format.
-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.

8 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.

9 References

1. *Introducing the Microsoft Vista event log format, by Andreas Schuster, 2007*
2. *Wikipedia, the free encyclopedia. Event Viewer topic*
3. *TechNet, New Tools for Event Management in Windows Vista*
4. *Randy Franklin Smith's online encyclopedia.*
5. *Windows Event Log Viewer, evtx_view, https://tzworks.com/prototype_page.php?proto_id=4*
6. *SleuthKit Body-file format, http://wki.sleuthkit.org/*
7. *Log2timeline CSV format, http://log2timeline.net/*
8. *SQLite library statically linked into tool [Amalgamation of many separate C source files from SQLite version 3.32.3].*
9. *SQLite documentation [http://www.sqlite.org].*
10. *DB Browser for SQLite [http://sqlitebrowser.org/]*