TZWorks® Timeline ActivitiesCache Parser (*tac*) Users Guide

Abstract

tac is a standalone, command-line tool that parses the Windows Timeline records introduced in the April 2018 Win10 update. The Window Timeline functionality makes use of an SQLite database to store which applications have been run.

Copyright © TZWorks LLC
www.tzworks.com
Contact Info: info@tzworks.com
*Document applies to v0.35 of **tac***
Updated: Apr 25, 2025

Table of Contents

TZWorks® Timeline ActivitiesCache Parser (*tac*) Users Guide

Copyright © *TZWorks LLC*
Webpage: http://www.tzworks.com/prototype_page.php?proto_id=41
Contact Information: info@tzworks.com

1   Introduction

In the spring of 2018, Microsoft released a Win10 update (version 1803) with the capability to show a chronology of actions taken by the user. This new application is called Timeline and is part of Windows Task View. It allows one to go back in time to find the items previously worked on. It has a history from the most recent tasks to a few weeks ago (up to 30 days).   Whether going back to a previous Internet search done some time ago, or continuing on with the document that was been read or edited, the functionality is built into the Timeline application to track this.

For the forensic analyst, since this service is turned on by default, the amount of data that can be collected with an account's activity is very useful. To disable the capability, one would need to explicitly turn it off in the Privacy settings.
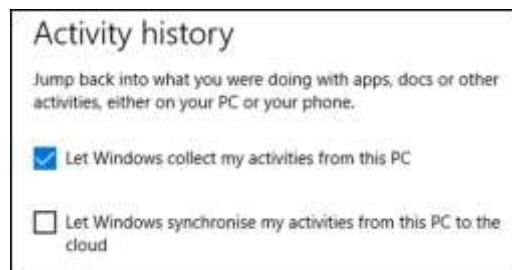
If the activity history was not turned off, then the analyst can retrieve items, such as: which file was viewed and/or edited, website visited, corresponding timestamps, etc.

The database storing the user's activity is the *ActivitiesCache.db*.   Each user account has its own database, and it can be found in one of these locations:
C:\Users\<useracct>\AppData\Local\ConnectedDevicesPlatform\L.<useracct>\ActivitiesCache.db.
C:\Users\<useracct>\AppData\Local\ConnectedDevicesPlatform\<cid>\ActivitiesCache.db.

The first on listed is for accounts that are locally logged into; whereas the second is for online accounts.

Below is a truncated output of looking into the records of the *ActivitiesCache.db*. One can see when the application was run and how long it was running. The expiration time is something that allows the timeline to only keep those items on the list that are within a set amount of time to keep the timeline of items manageable.   There are many other fields that are used in the database that are not shown

below; many of them still need to be studied to determine what they are and if they are of forensic value.

Application	Path	ActivityType	LastModifiedTime	ExpirationTime	ActiveDurationSecs
yaru64.exe	C:\tools\home	5	07/12/2018 15:26:25	08/11/2018 15:26:25	
yaru64.exe	C:\tools\home	6	07/12/2018 15:26:44	08/11/2018 15:26:44	55
Wireshark.exe	{CLSID_ProgramFiles}\Wireshark	5	07/15/2018 16:19:18	08/14/2018 16:19:18	
Wireshark.exe	{CLSID_ProgramFiles}\Wireshark	6	07/15/2018 16:19:48	08/14/2018 16:19:48	26
010Editor.exe	{CLSID_ProgramFiles}\010 Editor	5	07/19/2018 21:27:46	08/18/2018 21:27:46	
010Editor.exe	{CLSID_ProgramFiles}\010 Editor	6	07/19/2018 21:27:57	08/18/2018 21:27:57	11
VisualStudio.10.0		5	07/19/2018 21:32:25	08/18/2018 21:32:25	
VisualStudio.10.0		6	07/19/2018 21:32:25	08/18/2018 21:32:25	177

The *ActivitiesCache.db* has a number of tables, many of which are used for indexing and fast searching. The **tac** tool targets only two of these database tables; the *Activity* and the *ActivityOperation* tables. Collectively, they contain all the information needed to reconstruct which applications were run. Depending on the version of Windows 10 or Windows 11 the database schema has more or less fields. The schema in Windows version 23H2 is shown below.

Activity	
Id	GUID
AppId	TEXT
PackageIdHash	TEXT
AppActivityId	TEXT
ActivityType	INT
ActivityStatus	INT
ParentActivityId	GUID
Tag	TEXT
Group	TEXT
MatchId	TEXT
LastModifiedTime	DATETIME
ExpirationTime	DATETIME
Payload	BLOB
Priority	INT
IsLocalOnly	INT
PlatformDeviceId	TEXT
DdsDeviceId	TEXT
CreatedInCloud	DATETIME
StartTime	DATETIME
EndTime	DATETIME
LastModifiedOnClient	DATETIME
GroupAppActivityId	TEXT
ClipboardPayload	BLOB
EnterpriseId	TEXT
OriginalPayload	BLOB
UserActionState	INT
IsRead	INT
OriginalLastModifiedOnClient	DATETIME
GroupItems	TEXT
LocalExpirationTime	DATETIME
ETag	INT

ActivityOperation	
OperationOrder	INTEGER
Id	GUID
OperationType	INT
AppId	TEXT
PackageIdHash	TEXT
AppActivityId	TEXT
ActivityType	INT
ParentActivityId	GUID
Tag	TEXT
Group	TEXT
MatchId	TEXT
LastModifiedTime	DATETIME
ExpirationTime	DATETIME
Payload	BLOB
Priority	INT
CreatedTime	DATETIME
OperationExpirationTime	DATETIME
PlatformDeviceId	TEXT
DdsDeviceId	TEXT
CreatedInCloud	DATETIME
StartTime	DATETIME
EndTime	DATETIME
LastModifiedOnClient	DATETIME
CorrelationVector	TEXT
GroupAppActivityId	TEXT
ClipboardPayload	BLOB
EnterpriseId	TEXT
UserActionState	INT
IsRead	INT
OriginalPayload	BLOB
OriginalLastModifiedOnClient	DATETIME
UploadAllowedByPolicy	INT
PatchFields	BLOB
GroupItems	TEXT
ThrottleReleaseTime	DATETIME
PublishProcessStatus	INT
ETag	INT

As Windows 10 evolved from v1803 up to later versions and then Windows 11, the schema has changed as well. Each new version of Windows included more fields. The *tac* tool is aware of the schema versions that exist in Win10 and Win11 (specifically from v1803, v1809, v1903, v21H1 through 23H2).

Included in each of the table's metadata are various timestamps, the application (and sometimes the path it was run from), the duration the application was in use, etc. While there is some data not understood by this developer, there is plenty of useful critical data readily identifiable, such as: when a specific application was run, by which user account, and if there were any associated documents/pages used with the application. As more meaningful data is discovered, this tool will continue to evolve accordingly.
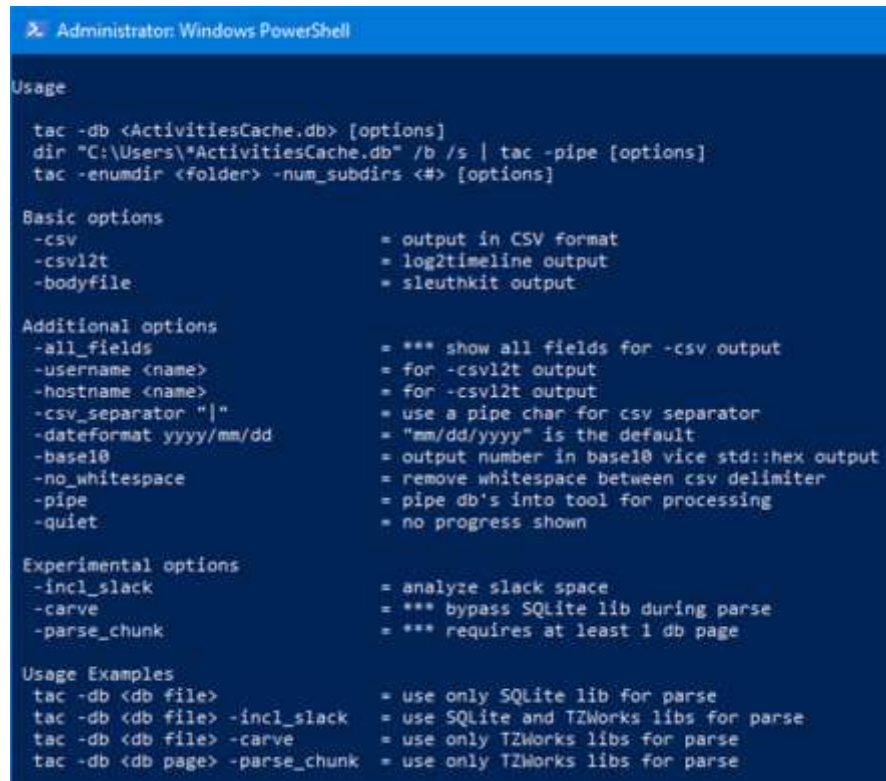
Empirical testing shows that as an application is run, typically at least 2 entries in the *Activity* Table are created (one with an *ActivityType* 5 and one with an *ActivityType* 6). What each number represents is just a guess, but from the data thus far, it suggests that *ActivityType* 5 is the initial type representing when the application started, and the *ActivityType* 6 is the ending type representing when the application ended. The *ActivityType* 6 records the duration time (*ActiveDurationTime*). At this point, these conclusions are preliminary and are subject to change as more data is analyzed. For the purposes of completeness, the *tac* tool outputs both entries in the output.

Empirical testing also shows that when the user manually removes an item from the Windows Timeline, the *ActivityOperation* Table records that action as well as the time this occurred via the *CreatedTime* field.

Also, noteworthy, is the Windows Timeline doesn't necessary display the application that was run within the timeline, however, the application does get recorded in the *ActivitiesCache* database. Whether the display inconsistency is due to the initial release of the Windows Timeline and whether Microsoft may be focusing on Windows applications over 3rd party applications is unknown at this time. But since the data does seem to be captured into the *ActivitiesCache* database, this artifact is very useful from a forensics standpoint.

2   How to Use *tac*

The screen shot below shows all the options available.

The semantics to run this tool just requires one to use the **-db** option and pass in the path/file of the *ActivitiesCache.db* to parse. The parsed output will dump to the screen, unless one redirects the output to a file, as shown below:

The above command will only parse the more commonly used fields from the *Activity* and *ActivityOperation* tables. To parse all the fields from those same tables, one can use the **-all_fields** option in the command above.

2.1 Recovering Discarded Records

When considering how to recover records that have been deleted or overwritten, the SQLite database can be treated as a file system; as such, one can analyze which pages in the database are valid and which are invalid. Of the valid pages, one can look into those areas that are slack space. Finally, one can look into the journaling file and see which areas also can be considered for record recovery.   Identifying these unused areas (or areas marked as invalid), one can look at the data available and see if records of interest can be extracted. *tac* has the *-incl_slack* option to do this. This will cause the tool to: (a) traverse the database, and if available, the journaling file as well, (b) identify which areas are invalid or which are slack space and (c) look for records that are of interest and extract them.   From the empirical testing on various sample database files, this option has yielded a sizable number of records when compared to the just extracting the valid records.

It goes without saying, that anytime a parser tries to reconstruct records from corrupted (partially overwritten) data, it should be cause for concern. That is why this option is still experimental. Consequently, using this option could cause the parser to crash if it tries to reconstruct a record that causes an *out-of-bound* condition in the internal parsing algorithm. More datasets are required with differing boundary conditions for this option in the tool to come out of the experimental category.

Even though the extra output produced by the *-incl_slack* option will include additional records, one should be aware, that a recovered record is not necessarily a 'deleted' record, but can be copied version of an older record (and it may be identical). To provide additional tracking information, *tac* will output any additional metadata in a separate column called "*misc data*"; a sample output of this field is shown below:

The above metadata will identify where the recovered record came from, as well as the offset into the file. If a record was truncated, that will also be annotated. If the database was upgraded to a newer schema, there may also be records from the older database schema present. From the example above, the shaded blue areas are recovered records prior to the database upgrading from Win10 v1803 to v1809. Collectively, this should be enough data to give the investigator the information to manually review the raw data in a hex viewer for verification purposes.

2.2 Processing Multiple Databases

If desiring to process many database files in one pass, one can put the artifact database in separate subdirectories that share a common parent folder (or just enumerate them on a live system) and use the *-pipe* option like so:

This will process all the databases and output the results into one file. To help distinguish which lines go to which database file, an extra field is appended to each record identifying the source database.

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

2.3 Bypassing the Embedded SQLite library

The *tac* tool has the SQLite library embedded into the binary. More information about this is discussed in the section *Use of the SQLite Library*. Sometimes, however, one may not wish to use the SQLite library for analyzing the records, so an option was added to bypass the SQLite library and use the *TZWorks'* internal SQLite algorithms to parse the database. This functionality can be invoked one of two ways: (a) with the *-carve* option or (b) the *-parse_chunk* option. Out of the two options, one should opt for the first, *-carve* option first. This option will try to traverse the database (even corrupted ones) and should pull out all the same information as if using the normal SQLite library plus recover any records in the discarded pages. The difference here is the *-carve* option is more immune to database corruption than the SQLite library is.

The purpose for the second option, *-parse_chunk*, is to go a step further and operate on only a subset of the database. More specifically, if at least a page of the database is available, this option will try to pull out any *Activity* and *ActivityOperation* records it finds. The limitation of this option is the data is segmented on page boundaries; this means any data crossing into other pages will be truncated.

However, if the data is not segmented on page boundary, but contiguous (as when reading the data from a file), then this option should have less truncation.

3   Use of the SQLite library

The *ActivitiesCache.db* is a SQLite database.   For the purposes of the *tac* tool, we statically link in the SQLite library to ensure the tool has minimal dependencies. The source code for the SQLite library is an amalgamation of the SQLite 'C' source files, version 3.32.3. More information about SQLite, the documentation and the source code can be seen at the official SQLite website [http://www.sqlite.org/].

Normally when we build a tool to parse a raw artifact, we prefer not to use outside libraries, however, in this case, the SQLite library has an option to open a SQLite database in '*read-only'* mode. From the testing done and from the documentation, it appears this is acceptable for this release. As an experimental option, we incorporated our internal library to assist parsing the slack and free space of a SQLite database. This gives the *tac* tool the ability to recover additional records that have been deleted or overwritten.   More testing in this area needs to be done to ensure the record recovery is robust to malformed records that have been overwritten.

4   Supporting Artifact files

The SQLite architecture uses a transactional system to ensure all database commits are done in an atomic fashion. This is implemented via the use of temporary files to allow the system to roll back to a previous state should the database abruptly quit in the middle of a transaction due to an unexpected power failure or a crash of the application controlling the SQLite database. The supplemental files used for this database are: *ActivitiesCache.db-wal* and *ActivitiesCache.db-shm.*

The first supplemental file is the 'write-ahead' log (WAL) and the second is the 'shared memory' file (SHM). The WAL file acts as a buffer for new records or changes to existing records to be recorded prior to flushing them to the database. This is the supplemental file of interest of the two from a forensics standpoint. The WAL file, by its nature, can grow very large since it can record many transactions prior to flushing them to the database.

Below is a screenshot of the user account *testuser*. One can see these files present below. Just looking at the size of the WAL file, which in this case is over 50% of the normal *ActivitiesCache.db* file, it should be clear that there is a significant amount of data contained in the file that has either been flushed to or is waiting to be flushed into the database.
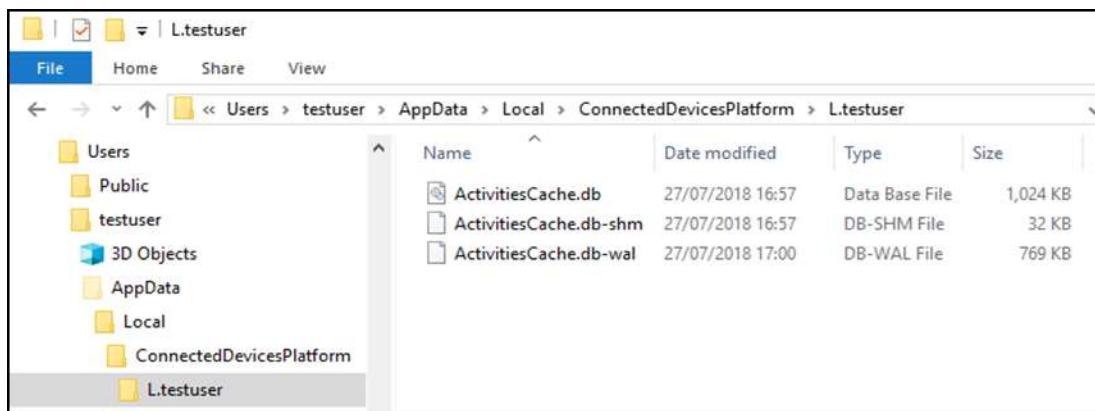
From a forensics standpoint, this offers the investigator additional data to analyze. Not only is the current record present, but potentially the previous version of the record as well. This occurs when the WAL contains the latest record prior to the commit (flush) while the database contains the older record.

In addition to the above behavior, it is common for the WAL file to save more than one transaction prior to the overall commit (flush to the database). Unfortunately, there is not a simple SQLite query to pull out invalid older records or examination of slack space. To build up this history, consisting of previous records, one needs to reconstruct them. This requires traversing the SQLite file internals to locate and extract invalid records, data in slack space, etc, and then try to match the data to the proper table schema available.

5   Timestamps

The timestamps stored in the *ActivitiesCache.db* are atypical for Windows. Normally the Windows operating system uses FILETIME to record timestamp data. In some cases, other timestamps like OLETIME will be used. In either case, the typical timestamps include a resolution down to the fractional seconds. This is not the case with the timestamps stored in the *ActivitiesCache.db*. The times stored here are expressed in 4-byte *time_t* type format (which is commonly called *Unix* time).   For this particular format, the timestamp resolution is in seconds (as opposed to fractional seconds) and as such, the *tac* tool has no need for a *-timeformat* option (which is used in other TZWorks tools), since the only purpose for *-timeformat* is to express time in varying fractional second resolutions.

From the two tables that *tac* is analyzing, there are at least 7 different timestamps available for the *Activity* table and 8 for the *ActivityOperation* table.

6 Available Options

Option	Description
-db	Specifies which database file to act on. The format is: *-db <activitiescache db to parse>*
-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 outputs are displayed in base-10 format versus hexadecimal format. Default is hexadecimal format.
-all_fields	Show all fields available for CSV output
-username	Option is used to populate the output records with a specified username. The format is: *-username <name to use>.*
-hostname	Option is used to populate the output records with a specified hostname. 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 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.
-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.
-quiet	Show no progress during the parsing operation.
-incl_slack	Experimental option to look at slack space free blocks to see if any records are present.
-carve	Experimental option. Bypass the SQLite embedded library and parse using TZWorks internal algorithms. This is for the situation where the database to be parsed is corrupted and the SQLite library has trouble parsing it.
-parse_chunk	Experimental option. Given a portion of the database (at least 1 page), this option will examine the data to see if any records exist and parse out the contents.
-utf8_bom	All output is in Unicode UTF-8 format. If desired, one can prefix an UTF-8 *byte order mark* to the output using this option.

7   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.

8   References

1. SQLite library statically linked into tool [Amalgamation of many separate C source files from SQLite version 3.32.3].
2. SQLite documentation [http://www.sqlite.org].