TZWorks® Windows Push Notification (wpn) Database Parser Users Guide

Abstract

wpn is a standalone, command-line tool that parses the Windows Push Notification (wpn) database. This database makes use of the SQLite architecture to store its records. This tool can parse both valid records and those that have been discarded or deleted (that have not be overwritten).

Copyright © TZWorks LLC

www.tzworks.net

Contact Info: info@tzworks.net

*Document applies to v0.13 of **wpn***

Updated: Jan 4, 2019

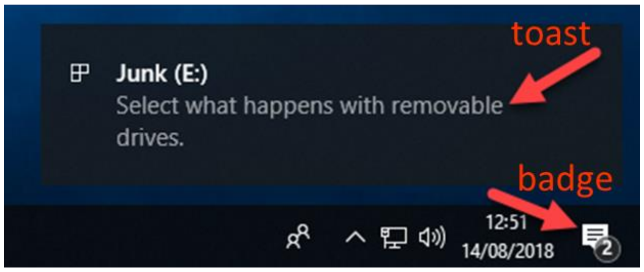
Table of Contents

TZWorks® Windows Push Notification (wpn) Database Parser Users Guide

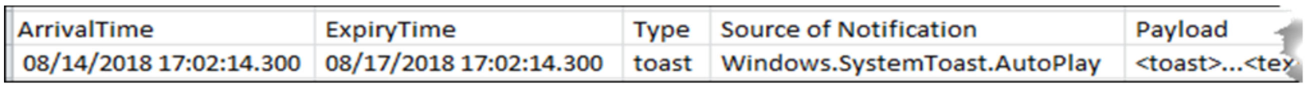
Webpage: http://www.tzworks.net/prototype_page.php?proto_id=42
Contact Information: info@tzworks.net

1 Introduction

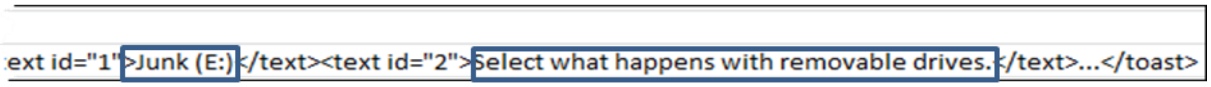
Starting in Windows 8, Microsoft created the Windows Push Notification Services (WNS) to allow applications to send *tile, toast, badge* and *raw updates*. The *tiles* are updates on application tiles such asweather updates, stock updates, etc. The *toasts* are another word for popups that occur, for example, when a new device is plugged into the computer requesting what action to take place. The *badges* are small overlays on the *tiles* on the desktop used to show the status or act as an active counter. Below is example of the operating system issuing a '*toast'* that a new device was attached to the computer. The icon on the bottom right with the number 2 is an example of a '*badge'* displaying the '2' value to show the Action Center has 2 new messages.



When looking at the internals of the *toast* and examining what data is captured, one can see the time the notification was sent out and the message that was displayed to the user. In this particular case the *toast* notification was not persistent as a valid record in the database after the toast was acted on; for this case, the ***wpn*** tool was able to pull out the *toast* record from slack space.



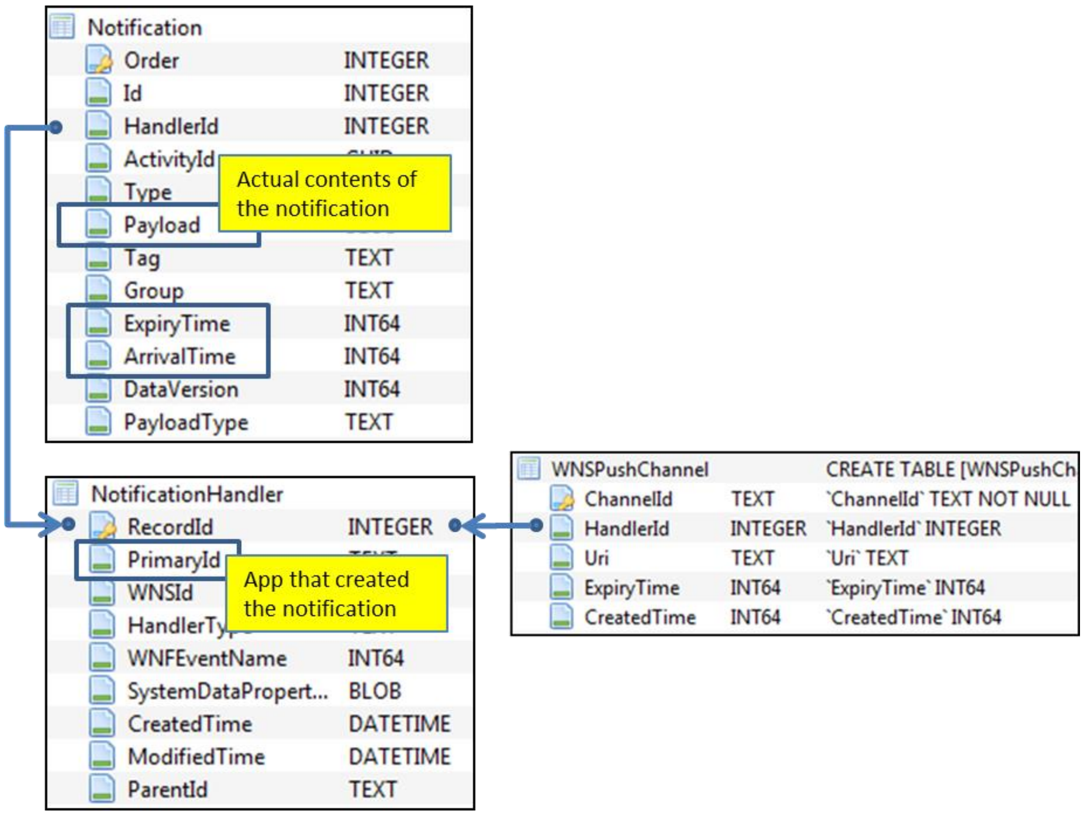
ArrivalTime	ExpiryTime	Type	Source of Notification	Payload
08/14/2018 17:02:14.300	08/17/2018 17:02:14.300	toast	Windows.SystemToast.AutoPlay	<toast>...<tex



ext id="1">Junk (E:)</text><text id="2">Select what happens with removable drives.</text>...</toast>

Later versions of Windows 10 made changes to the internal store format of the notification records. The newer updated format makes use of the SQLite architecture to store the data. Similar to the older style database, each user account has its own database instance to record the users notifications; for newer Windows 10, it can be found in this location:
C:\Users\<useracct>\AppData\Local\Microsoft\Windows\Notifications\wpndatabase.db.

The ***wpn*** tool only targets the newer format of the notifications in Windows 10 and not the older format that was used.
The *wpndatabase.db* has a number of tables. From these tables, ***wpn*** looks at most of them, but primarily targets two of them for the bulk of the information; the *Notification* and *NotificationHandler* tables. Collectively, they contain the requisite information to determine the application that initiated the notification, content of the notification, and when it was posted. The schema (or fields) used for these tables are shown below.

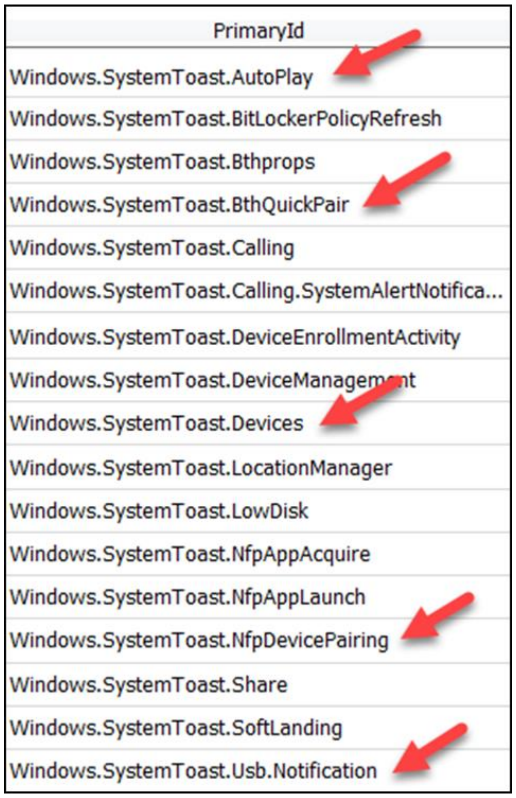

Actual contents of the notification
App that created the notification

Notification	
Order	INTEGER
Id	INTEGER
HandlerId	INTEGER
ActivityId	GUID
Type	
Payload	
Tag	TEXT
Group	TEXT
ExpiryTime	INT64
ArrivalTime	INT64
DataVersion	INT64
PayloadType	TEXT

NotificationHandler	
RecordId	INTEGER
PrimaryId	TEXT
WNSId	
HandlerType	
WNFEventName	INT64
SystemDataPropert...	BLOB
CreatedTime	DATETIME
ModifiedTime	DATETIME
ParentId	TEXT

WNSPushChannel		CREATE TABLE [WNSPushCh
ChannelId	TEXT	`ChannelId` TEXT NOT NULL
HandlerId	INTEGER	`HandlerId` INTEGER
Uri	TEXT	`Uri` TEXT
ExpiryTime	INT64	`ExpiryTime` INT64
CreatedTime	INT64	`CreatedTime` INT64

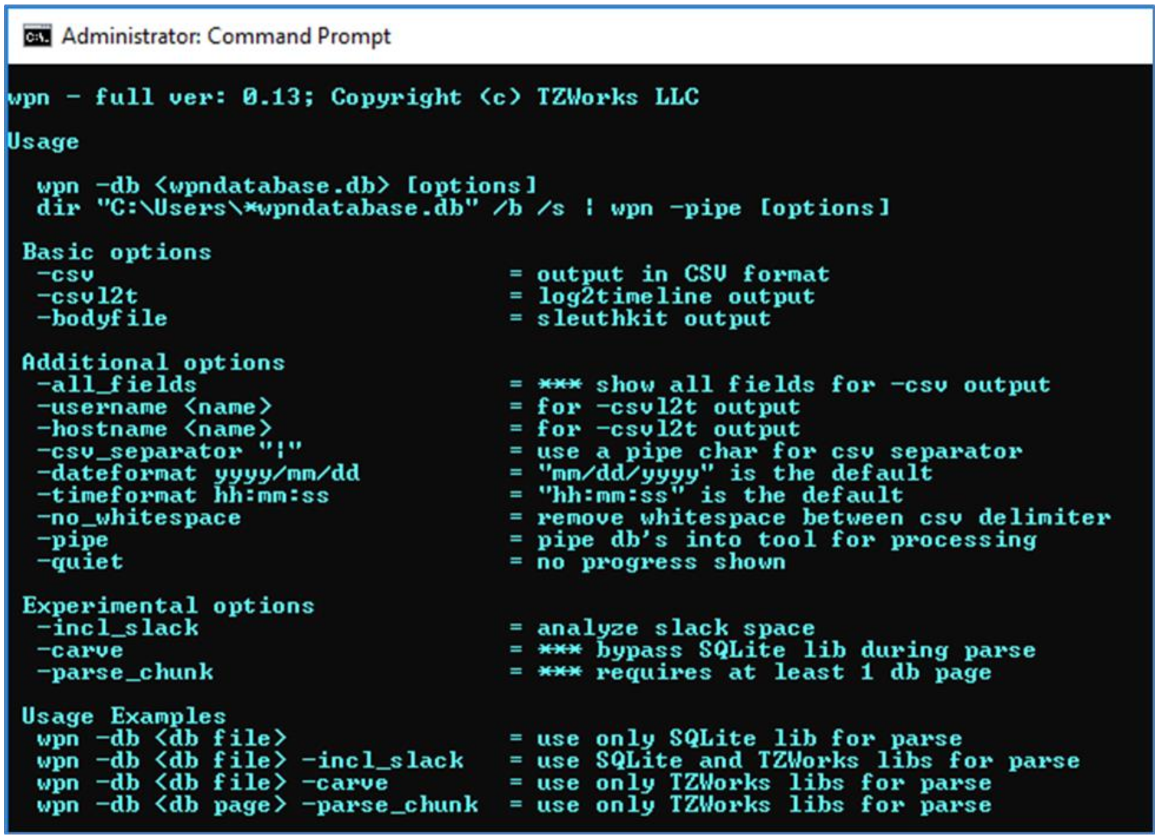
Since the database uses a relational type architecture, there are references from one table to other table(s). Starting with the *Notification* table, the *HanderId* field in each record references an entry in the *NotificationHandler* table. By following these references one can recreate the final record for the analyst to review. Highlighted above are some of the more useful fields. One of these fields is the *PrimaryId* in the *NotificationHandler* table. Just a quick look through some of the entries created by the operating system by default, include some interesting artifacts that relate to devices as shown below. There are entries for: *Autoplay*, *BlueTooth pairing, Devices* (in general) and *USB devices* (in particular). Other entries, not shown are entries for specific applications that have registered to push out notifications.



PrimaryId
Windows.SystemToast.AutoPlay
Windows.SystemToast.BitLockerPolicyRefresh
Windows.SystemToast.Bthprops
Windows.SystemToast.BthQuickPair
Windows.SystemToast.Calling
Windows.SystemToast.Calling.SystemAlertNotifica...
Windows.SystemToast.DeviceEnrollmentActivity
Windows.SystemToast.DeviceManagement
Windows.SystemToast.Devices
Windows.SystemToast.LocationManager
Windows.SystemToast.LowDisk
Windows.SystemToast.NfpAppAcquire
Windows.SystemToast.NfpAppLaunch
Windows.SystemToast.NfpDevicePairing
Windows.SystemToast.Share
Windows.SystemToast.SoftLanding
Windows.SystemToast.Usb.Notification

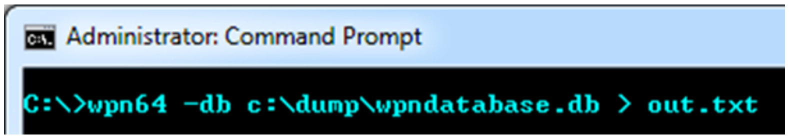
2 How to Use *wpn*

The screen shot below shows all the options available.


Administrator: Command Prompt

```
wpn - full ver: 0.13; Copyright (c) TZWorks LLC
Usage

  wpn -db <wpndatabase.db> [options]
  dir "C:\Users\*wpndatabase.db" /b /s | wpn -pipe [options]

 Basic options
  -csv                              = output in CSV format
  -csvl2t                           = log2timeline output
  -bodyfile                         = sleuthkit output

 Additional options
  -all_fields                       = *** show all fields for -csv output
  -username <name>                  = for -csvl2t output
  -hostname <name>                  = for -csvl2t output
  -csv_separator "|"                = use a pipe char for csv separator
  -dateformat yyyy/mm/dd            = "mm/dd/yyyy" is the default
  -timeformat hh:mm:ss              = "hh:mm:ss" is the default
  -no_whitespace                    = remove whitespace between csv delimiter
  -pipe                             = pipe db's into tool for processing
  -quiet                            = no progress shown

 Experimental options
  -incl_slack                       = analyze slack space
  -carve                            = *** bypass SQLite lib during parse
  -parse_chunk                      = *** requires at least 1 db page

 Usage Examples
  wpn -db <db file>                 = use only SQLite lib for parse
  wpn -db <db file> -incl_slack     = use SQLite and TZWorks libs for parse
  wpn -db <db file> -carve          = use only TZWorks libs for parse
  wpn -db <db page> -parse_chunk    = use only TZWorks libs for parse
```

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


Administrator: Command Prompt

```
C:\>wpn64 -db c:\dump\wpndatabase.db > out.txt
```

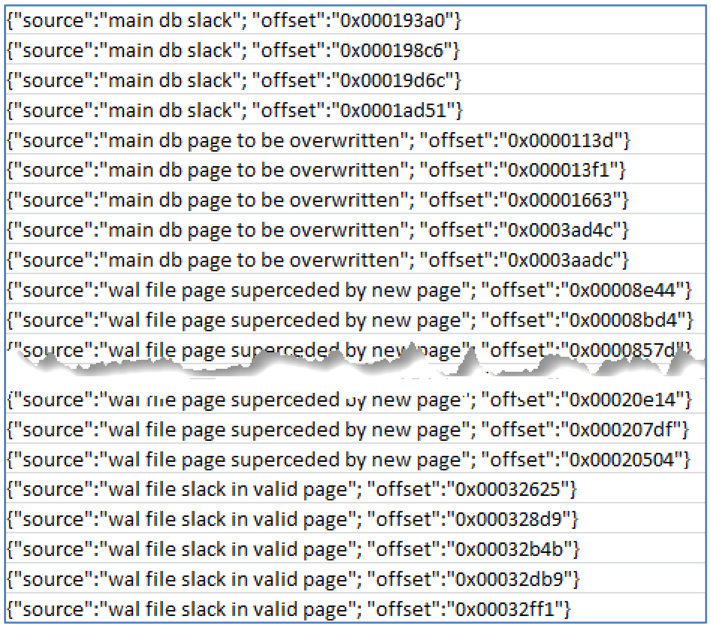
The above command will only parse the more commonly used fields from the *Notification* and *NotificationHandler* tables. To parse all the fields from those same tables, one can use the ***-all_fields*** option in the command above.

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. ***wpn*** 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 slack space and (c) look for records that are of interest and extract them. From the empirical testing on sample *wpndatabase.db* files, this has yielded a sizable number of records when compared to the valid records normally returned without using this option.

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. 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 older record (and it may be identical). To provide additional tracking information, ***wpn*** will output any additional metadata in a separate column called "*misc data*"; a sample output of this field is shown below:

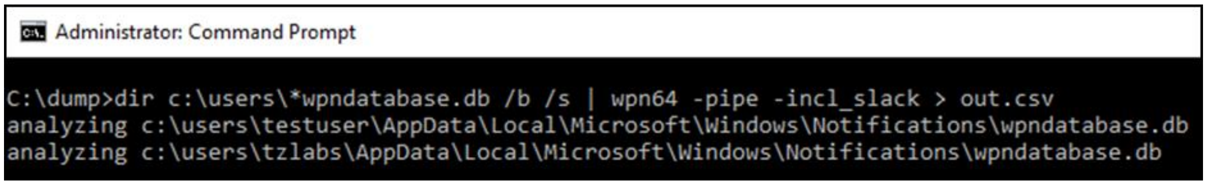


```
{"source":"main db slack"; "offset":"0x000193a0"}
{"source":"main db slack"; "offset":"0x000198c6"}
{"source":"main db slack"; "offset":"0x00019d6c"}
{"source":"main db slack"; "offset":"0x0001ad51"}
{"source":"main db page to be overwritten"; "offset":"0x0000113d"}
{"source":"main db page to be overwritten"; "offset":"0x000013f1"}
{"source":"main db page to be overwritten"; "offset":"0x00001663"}
{"source":"main db page to be overwritten"; "offset":"0x0003ad4c"}
{"source":"main db page to be overwritten"; "offset":"0x0003aadc"}
{"source":"wal file page superceded by new page"; "offset":"0x00008e44"}
{"source":"wal file page superceded by new page"; "offset":"0x00008bd4"}
{"source":"wal file page superceded by new page"; "offset":"0x0000857d"}
{"source":"wal file page superceded by new page"; "offset":"0x00020e14"}
{"source":"wal file page superceded by new page"; "offset":"0x000207df"}
{"source":"wal file page superceded by new page"; "offset":"0x00020504"}
{"source":"wal file slack in valid page"; "offset":"0x00032625"}
{"source":"wal file slack in valid page"; "offset":"0x000328d9"}
{"source":"wal file slack in valid page"; "offset":"0x00032b4b"}
{"source":"wal file slack in valid page"; "offset":"0x00032db9"}
{"source":"wal file slack in valid page"; "offset":"0x00032ff1"}
```

The above metadata will identify where the recovered record came from, as well as the offset into the file. 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:



Administrator: Command Prompt

```
C:\dump>dir c:\users\*wpndatabase.db /b /s | wpn64 -pipe -incl_slack > out.csv
analyzing c:\users\testuser\AppData\Local\Microsoft\Windows\Notifications\wpndatabase.db
analyzing c:\users\tzlabs\AppData\Local\Microsoft\Windows\Notifications\wpndatabase.db
```

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.

2.3 Bypassing the Embedded SQLite library

The ***wpn*** 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. 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 *Notification* records it finds. The limitations of this option include: (a) the data will need to start on a page boundary, and (b) it will not be able to provide joins between tables that have a relational aspect. Said another way, it will not combine the *NotificationHandler* table data with the *Notification* table data when outputting the results. The ***-carve*** option discussed earlier, however, will perform the necessary joins between tables that dependencies between them.

3 Use of the SQLite Library

The *wpndatabase.db* is a SQLite database. For the purposes of the ***wpn*** 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 3.19.3. More information about SQLite, the documentation and the source code can be seen at the official SQLite website [http://www.sqlite.org/].

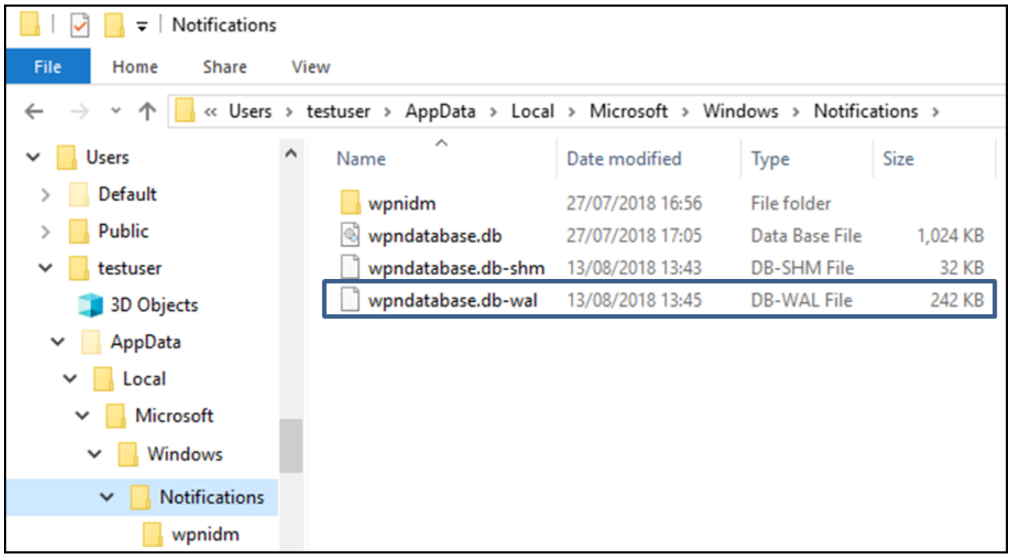
Normally when we build a tool to parse a raw artifact, we prefer not to use outside libraries, however, in this case, the SQLite library has an option to open a SQLite database in '*read-only*' mode. From the testing done and from the documentation, it appears this is acceptable for this release. As an experimental option, we incorporated our internal library to assist parsing the slack and free space of a SQLite database. This gives the ***wpn*** 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 rollback 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: *wpndatabase.db-wal* and *wpndatabase.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 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.



Name	Date modified	Type	Size
wpnidm	27/07/2018 16:56	File folder	
wpndatabase.db	27/07/2018 17:05	Data Base File	1,024 KB
wpndatabase.db-shm	13/08/2018 13:43	DB-SHM File	32 KB
wpndatabase.db-wal	13/08/2018 13:45	DB-WAL File	242 KB

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 (or Write Ahead Log) 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 Available Options

Option		Description
-db		Specifies which database file to act on. The format is: ***-db <wpndatabase.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.
-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.
-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		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 "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.
-timeformat		Output the time using the specified format. Default behavior is ***-timeformat "hh:mm:ss.xxx"*** One can adjust the format to microseconds, via ***"hh:mm:ss.xxxxxx"*** or nanoseconds, via ***"hh:mm:ss.xxxxxxxxx"***, or no fractional seconds, via ***"hh:mm:ss"***. The restrictions with this option is a colon (:) symbol needs to separate hours, minutes and seconds, a period (.) symbol needs to separate the seconds and fractional seconds, and the repeating symbol 'x' is used to represent number of fractional seconds.
-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.

Options with an asterisk require a commercial license to be used.

6 Authentication and the License File

This tool has authentication built into the binary. There are two authentication mechanisms: (a) the digital certificate embedded into the binary and (b) the runtime authentication. For the first method, only the Windows and Mac OS-X (if available) versions have been signed by an X-509 digital code signing certificate, which is validated by Windows (or OS-X) during operation. If the binary has been tampered with, the digital certificate will be invalidated.

For the second (runtime authentication) method, the authentication does two things: (a) validates that the tool has a valid license and (b) validates the tool's binary has not been corrupted. 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. The runtime binary validation hashes the executable that is running and fails the authentication if it detects any modifications.

7 References

1. Windows Push Notification Services (WNS) Overview; [https://docs.microsoft.com/en-us/windows/uwp/design/shell/tiles-and-notifications/windows-push-notification-services--wns--overview].
2. Badge Notifications for UWP apps; [https://docs.microsoft.com/en-us/windows/uwp/design/shell/tiles-and-notifications/badges].
3. Toast Content; [https://docs.microsoft.com/en-us/windows/uwp/design/shell/tiles-and-notifications/adaptive-interactive-toasts].
4. Raw Notification Overview; [https://docs.microsoft.com/en-us/windows/uwp/design/shell/tiles-and-notifications/raw-notification-overview].
5. SQLite library statically linked into tool [Amalgamation of many separate C source files from SQLite version 3.19.3].
6. SQLite documentation [http://www.sqlite.org].