TZWorks® Chromium SQLite Parser (*csp*) Users Guide

Abstract

csp is a standalone, command-line tool that parses certain databases associated with Chromium-based browsers (such as the browsers: Chrome, Edge, Opera, Brave and Vivaldi). This tool incorporates three separate parsing engines to facilitate operating in varying environments. Two of these engines can parse both valid and discarded records as well as parse records from corrupted SQLite databases. One of the parsing engines allows for parsing records from partial fragments of a database. This tool has working versions for Windows, Linux and OS-X.

Copyright © TZWorks LLC

www.tzworks.com

Contact Info: info@tzworks.com

*Document applies to v0.27 of **csp***

Updated: Apr 25 2025

Table of Contents

TZWorks® Chromium SQLite Parser (*csp*) Users Guide

Webpage: http://www.tzworks.com/prototype_page.php?proto_id=45
Contact Information: info@tzworks.com

1 Introduction

As background, the *Blink* engine is used in the current *Chromium* architecture. Other common browser engines include *Gecko* and *WebKit*. Below is a table to showing where the *Chromium* architecture is used, and consequently, which browsers and the respective SQLite tables, the ***csp*** tool targets.

Tool	Browser Type	SQLite Tables Targeted by Tool	Notes
csp	*Chromium* based that use the *Blink* engine (e.g. *Edge, Chrome, Brave, Vivaldi,* etc.)	*urls, visits, keyword_search_terms, visit_source, downloads, downloads_url_chains, clusters_and_visits, content_annotations, context_annotations, cookies, autofill, thumbnails, top_sites, omni_box_shortcuts, logins, favicons, favicon_bitmaps, nel_policies, bounces*	The ***csp*** tool just targets the SQLite data and the ***ccp*** tool is used to parse the cache
msp	*Mozilla* based that use the *Gecko* engine (e.g. *Firefox, SeaMonkey, Tor Browser*, etc.)	*moz_places, moz_origins, moz_bookmarks, moz_historyvisits, moz_inputhistory, moz_keywords, moz_annos, moz_items_annos, moz_anno_attributes, moz_cookies, moz_downloads, moz_icons, moz_icons_to_pages, moz_pages_w_icons, moz_favicons, moz_formhistory*	The ***msp*** tool just targets the SQLite data and the ***mcp*** tool is used to parse the cache
sap	*WebKit* based browsers (e.g. *Safari*)	*history_items, history_visits, history_items_to_tags, history_tags, icon_info, page_url, cache_settings, cloud_tabs, cloud_tab_devices, cfurl_cache_blob_data, cfurl_cache_receiver_data, cfurl_cache_response, ItemTable*	The ***sap*** tool also parses the cache, as well as, some *plists* containing useful data

As shown in the first row above, the Chromium architecture used in many browsers and has many artifacts available that the forensics examiner can use in identifying a user's Internet activity. This includes Chromium's various databases, local storage, JSON formatted text files, and its cache.

This tool, however, does not target all of Chromium's artifacts; it only targets certain SQLite databases and specific tables within those databases that are used by the Browser that have been deemed useful by the forensics community. Specifically, this tool targets the following seven databases: (a) History, (b) Cookies, (c) Web Data, (d) Top Sites, (e) Shortcuts, (f) Login Data, and (g) Favicons. Each of these databases will be discussed in turn.

The user should be aware that the database schemas (which identifies the fields and their respective types that comprise of a record) sometime vary across Chromium versions. This is something that needed to be taken into consideration when designing the ***csp*** tool. The solution chosen was to have the tool dynamically detect and adjust to varying schemas as they are encountered during the parsing operation. This makes for a more complex algorithm and therefore done on a best effort basis.

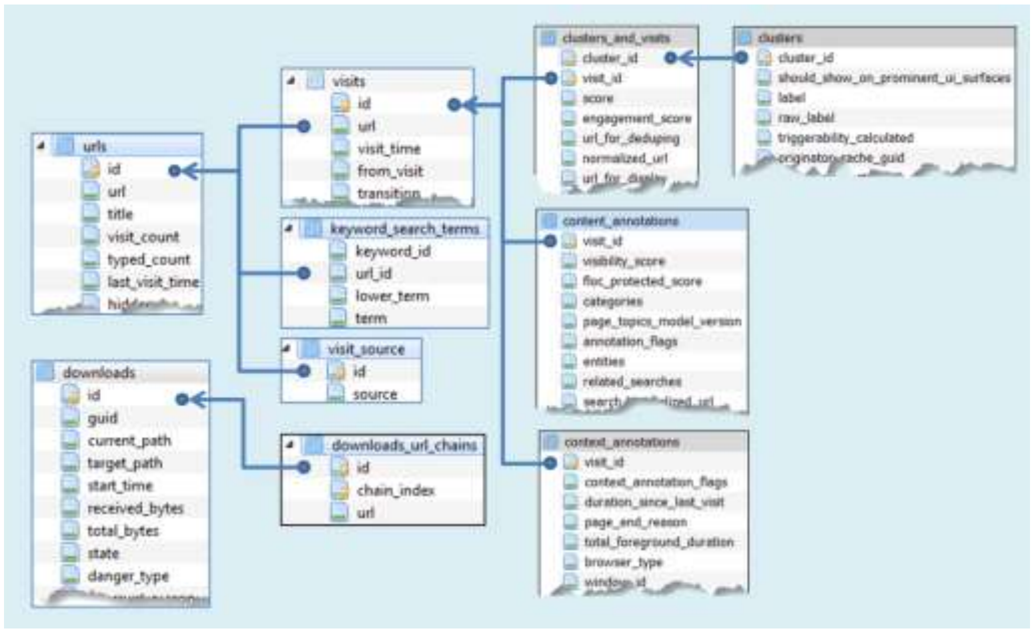
In addition to the auto-schema detection, the ***csp*** tool offers three ways to parse a target database. (1) The first way makes use of a standard SQL (Structured Query Language) parse. This is internal to the tool, so SQL knowledge from the user is not required. For this option, the SQLite library was statically linked into the tool, so a third-party dynamic library is not required to be present to run the tool. (2) The second approach tells the tool to traverse the internal SQLite structures and parse out the records as they are encountered. This option does not use the standard SQLite library, but utilizes the TZWorks' internally designed libraries. The benefit of 'rolling your own' library is multi-faceted; not only does it allow the tool to extract records from a corrupted database, but one can record the exact offset of the data where it was found, if desiring to validate it later with a hex-editor. (3) The third, and final approach, uses a *signature-based* parse. This turns out to be a unique way in parsing out records and the beauty of it is that it does not use the internal SQLite table structures. Consequently, this latter option allows one to parse records from a blob of data whether it be from memory or from a fragment of a database. All three approaches were left in the tool for the analyst to use since there are benefits, and disadvantages, for each option depending on the situation. More discussion on these options are discussed in a later section.

2 Databases Targeted by this tool

2.1 History Database

Chromium's ***History*** database has a number of tables of interest to the analyst. Below is a diagram of these tables and their relationship to each other. The fields listed in the tables below may or may not be valid on differing versions of Chromium due to varying schemas with later versions.

Aside from the *URL* data, one can associate *visit time*, whether something was *downloaded*, its *size*, whether it was *opened*, and the *current path* where it was stored.

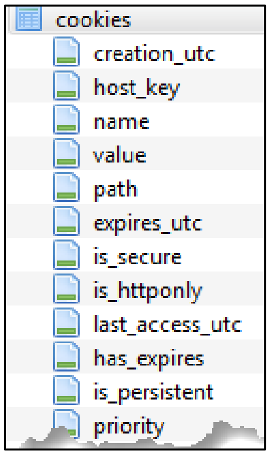


The table relationships shown by the line connectors above will have an effect on the number of records that will be outputted by the ***csp*** tool. For example, the tables *urls, visits, and keyword_search_terms* have what is called a '*one to many*' relationship. The *visits* and *keyword_search_terms* tables may have many linked records to only one entry in the *urls* table***.*** Therefore, after merging the data in the various tables together based on their relationship, one may get more records in the output of the report then the number of records in the *urls* table. This is because each parsed line in the output has taken the '*one to many*' relationship and converted it to a '*one to one'* relationship; where each line in the output shows one *url* entry and one *visit* entry. If there was a second *visit* entry for the same *url* entry, that would constitute a separate output line.

For the ***csp*** tool, each parsed line in the output shows one *url* entry and one *visit* unique entry. This is because the various timestamps recorded in the *visits* table can be digested better by other tools when broken out as a separate line per *visit* timestamp. Alternatively, the *keyword_search_terms* table also has a '*many to zero/one'* relationship with the *urls* table. However, in this case, since no timestamps are in the *keyword_search_terms* records, the keywords from all the matching records can be then merged into one output for each *url* entry.

2.2 Cookies Database

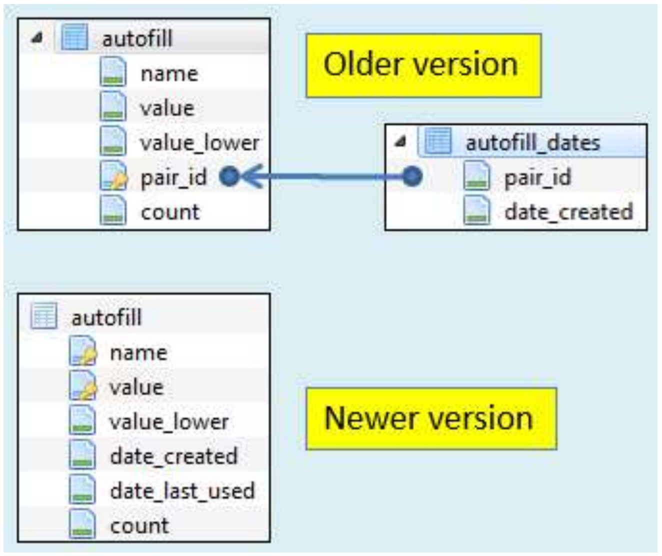
The ***Cookies*** database has some interesting data, including timestamps when it was created, last access time, and expiration. If the cookie is from Google, the *cookie value* contains additional timestamps and other miscellaneous information. The ***csp*** tool outputs one parsed line for each record in the *cookies* table.



cookies
- creation_utc
- host_key
- name
- value
- path
- expires_utc
- is_secure
- is_httponly
- last_access_utc
- has_expires
- is_persistent
- priority

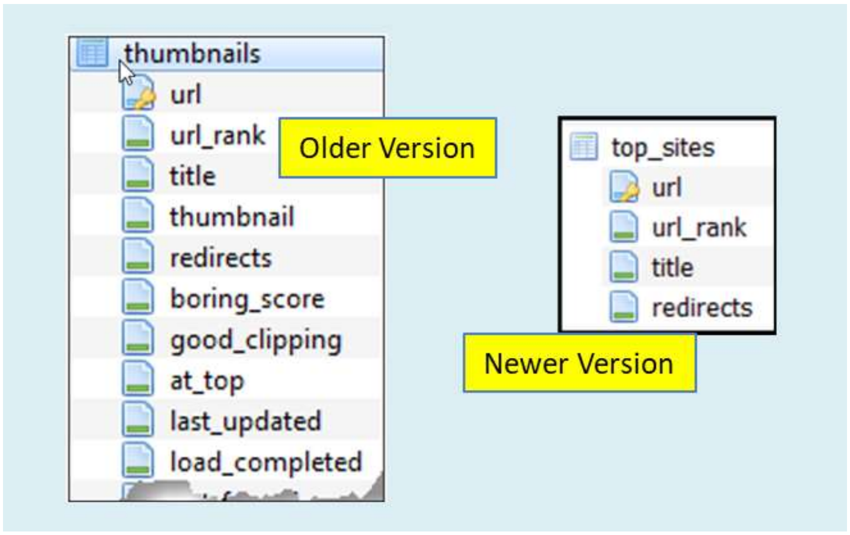
2.3 Web Data Database

The ***Web Data*** database has many tables, including *autofill*, *credit cards*, *phones numbers*, *names*, *keywords*, etc. The ***csp*** tool currently just parses records from the *autofill* table. Shown below are two versions of the *autofill* schema depending on the version the browser; the older version of Chromium uses two tables where the fields of both must be merged to get the *autofill name/value* along with the timestamp (*date created*). A later version of Chromium has all the required data to associate the *name/value* and *timestamps* together. The ***csp*** tool outputs one parsed line for each record in the *autofill* table.



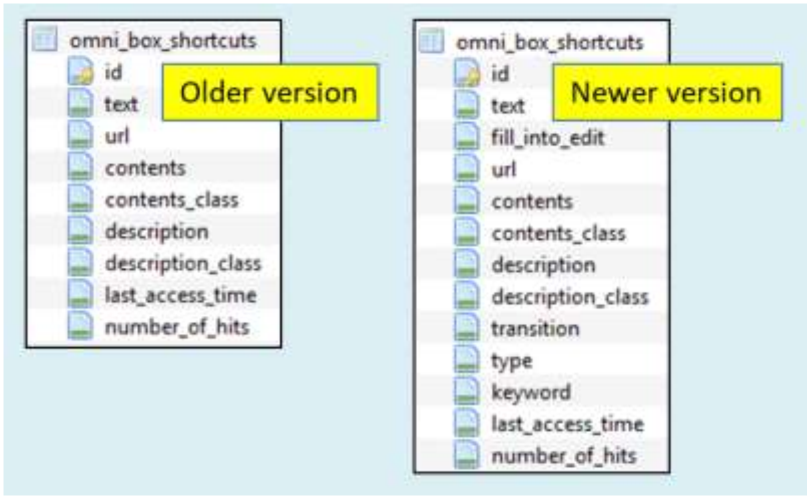
2.4 Top Sites Database

The ***Top Sites*** database has a table called *thumbnails*. This table includes the *URL* of the *thumbnail*, any *URL* redirects, and timestamps. The ***csp*** tool outputs one parsed line for each record in the *thumbnails* table for the older versions or *top_sites* in the newer versions.



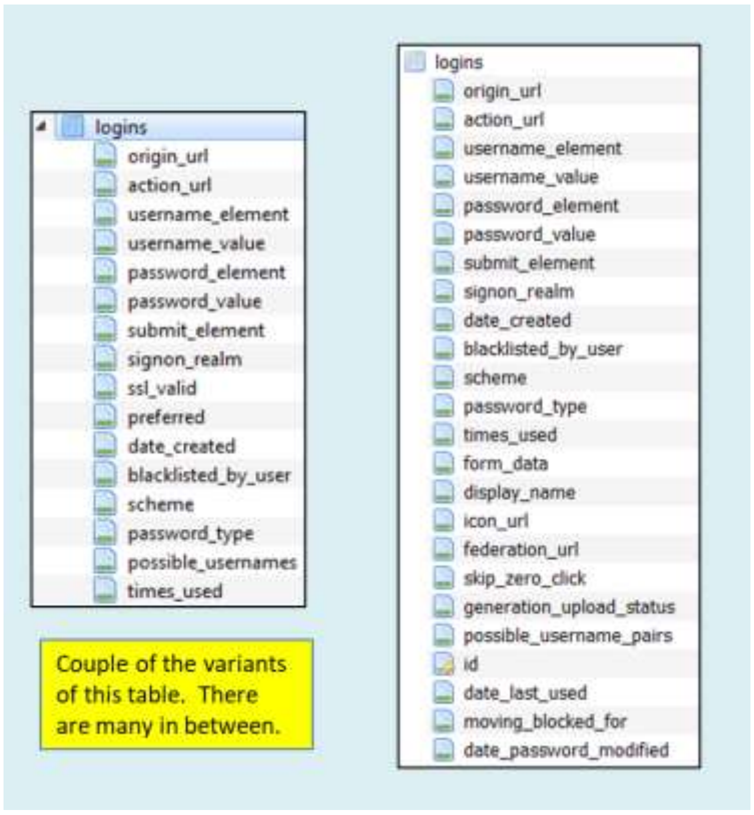
2.5 Shortcuts Database

The ***Shortcuts*** database has a table called *omni_box_shortcuts*. It has the *URL* of the site, *last access time*, and other metadata. Shown below are two versions of the *omni_box_shortcuts* schema depending on the version of Chromium/browser (*note*: there are other versions in between the two shown below). The ***csp*** tool outputs one parsed line for each record in the *omni_box_shortcuts* table.



2.6 Login Data Database

The ***Login Data*** database has the *logins* table. This contains the *URL* of the site, *date created*, the *username* and *password* (encrypted) and other information. This database Shown below are two variants of the table used in different versions of Chromium browsers. The ***csp*** tool outputs one parsed line for each record in the *logins* table.



```
logins
  origin_url
  action_url
  username_element
  username_value
  password_element
  password_value
  submit_element
  signon_realm
  ssl_valid
  preferred
  date_created
  blacklisted_by_user
  scheme
  password_type
  possible_usernames
  times_used
```

Couple of the variants of this table. There are many in between.

```
logins
  origin_url
  action_url
  username_element
  username_value
  password_element
  password_value
  submit_element
  signon_realm
  date_created
  blacklisted_by_user
  scheme
  password_type
  times_used
  form_data
  display_name
  icon_url
  federation_url
  skip_zero_click
  generation_upload_status
  possible_username_pairs
  id
  date_last_used
  moving_blocked_for
  date_password_modified
```

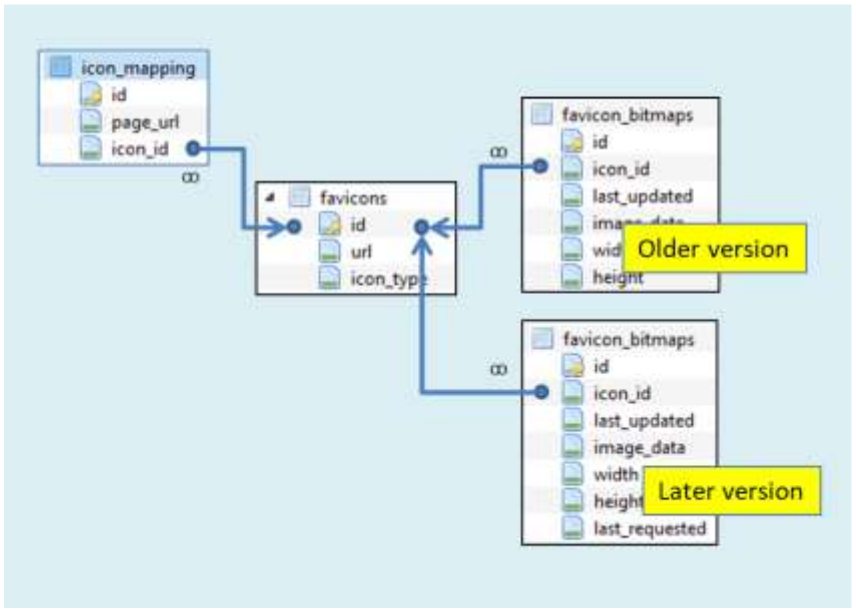
2.7 Favicons Database

The **Favicons** database contains a number of tables (shown below). The ones of interest are the *favicons, favicons_bitmap,* and the *icon_mapping* tables. From this data one can see the *url* (URL of the favicon*)*, the *last updated* timestamp, and the *url_page* (URL of the page visited). Shown below are two versions of the *favicon_bitmaps* schema depending on the version of Chromium; the older version of Chromium uses two subordinate tables to merge the fields to that of *favicons* data. The second one has one subordinate table that is merged with the *favicon* data.

Aside from the versioning differences, there is also a '*one to many*' relationship between these tables that needs to be addressed. For example, the *favicon_bitmaps* table can have multiple records for one *favicon* table entry (specifically the *favicon_bitmaps* will have a separate record for each resolution of the icon associated with the *favicon* table entry). Since each *favicon_bitmaps* entry has its own timestamp, these are looked at separately.

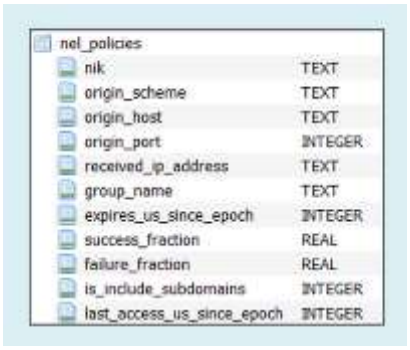
In addition, there is a '*one to many*' relationship when it comes to the *favicon* to *icon_mapping* tables. This means there can be many *icon_mapping* entries for each *favicon* entry.

For parsing to be successful, merging is required for at least the *favicon_bitmaps* to the *favicons* to get both the *URL* and the associated *last updated* timestamp. The additional parsing step is added to associate the *page url* to the results. The ***csp*** tool converts all the '*one to many*' or 'many to one' relationships by converting them to all '*one to one*' relationships in the output. Unfortunately, this makes for some verbose output. For traceability purposes, each line in the results is annotated with where the source data (by table) originated. This will help the analyst identify where all the data came from if validation is needed later on.



2.8 Reporting and NEL Database

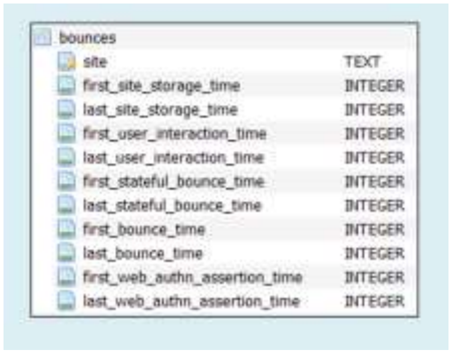
NEL is short for *Network Error Logging*. This is a mechanism for collecting *client-side* network errors. The ***nel_policies*** table have some interesting artifacts, including: a *last access* timestamp, an *expires* timestamp, *host URL*, *port number* and *received IP address*.



nel_policies	
nik	TEXT
origin_scheme	TEXT
origin_host	TEXT
origin_port	INTEGER
received_ip_address	TEXT
group_name	TEXT
expires_us_since_epoch	INTEGER
success_fraction	REAL
failure_fraction	REAL
is_include_subdomains	INTEGER
last_access_us_since_epoch	INTEGER

2.9 DIPS Database

The DIPS database is short for "*Detect Incidental Party State*". This is to track sites without any meaningful user interactions, such as bounce trackers. It does this by storing metrics in a "bounce" table which has been used as way for *server-side* websites to have a way to track browser activity without using Cookies.



bounces	
site	TEXT
first_site_storage_time	INTEGER
last_site_storage_time	INTEGER
first_user_interaction_time	INTEGER
last_user_interaction_time	INTEGER
first_stateful_bounce_time	INTEGER
last_stateful_bounce_time	INTEGER
first_bounce_time	INTEGER
last_bounce_time	INTEGER
first_web_authn_assertion_time	INTEGER
last_web_authn_assertion_time	INTEGER

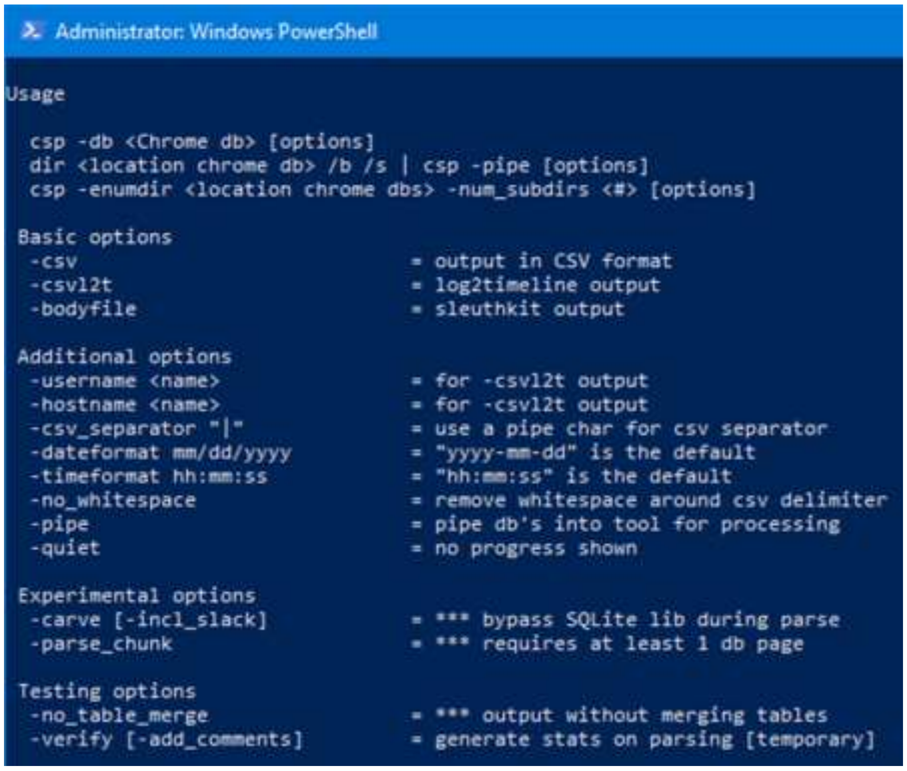
The 2 main categories of data include the: (a) site and (b) the various timestamps. The timestamps include when the user first/last interacted time, as well as other time data.

3 How to Use *csp*

Chromium artifacts are located in the user's directory. This varies depending on the operating system used.   Below is a table that breaks out the location by OS.

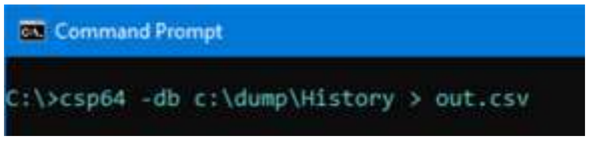
OS	Path
Win XP	%userprofile%\Local Settings\Application Data\Google\Chrome\User Data\Default
Post Win XP	%userprofile%\AppData\Local\Google\Chrome\User Data\Default %userprofile%\AppData\Local\Microsoft\Edge\User Data\Default %userprofile%\AppData\Local\BraveSoftware\Brave-Browser\User Data\Default %userprofile%\AppData\Local\Vivaldi\User Data\Default %userprofile%\AppData\Local\Opera Software\Opera Stable
OSX	/Users/{user}/Library/Application Support/Google/Chrome/Default /Users/{user}/Library/Application Support/Microsoft Edge/Default /Users/{user}/Library/Application Support/BraveSoftware/Brave-Browser/Default /Users/{user}/Library/Application Support/com.operasoftware.Opera/ /Users/{user}/Library/Application Support/Vivaldi/Default
Linux	/home/{user}/.config/google-chrome/Default /home/{user}/.config/Microsoft-edge/Default /home/{user}/.config/BraveSoftware/Brave-Browser/Default /home/{user}/snap/brave/[#]/.config/BraveBrowser/Brave-Browser/Default /home/{user}/snap/opera/[#]/.config/opera /home/{user}/.config/Vivaldi/Default

The semantics to run this tool simply requires one to use the ***-db*** option and pass in the path/file of the SQLite database to parse. The screenshot shows all the options available.



```
Usage

  csp -db <Chrome db> [options]
  dir <location chrome db> /b /s | csp -pipe [options]
  csp -enumdir <location chrome dbs> -num_subdirs <#> [options]

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

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

 Experimental options
  -carve [-incl_slack]              = *** bypass SQLite lib during parse
  -parse_chunk                      = *** requires at least 1 db page

 Testing options
  -no_table_merge                   = *** output without merging tables
  -verify [-add_comments]           = generate stats on parsing [temporary]
```

Below is an example of running the tool in its simplest form. Without explicitly setting any options, the tool will default to the *SQL Select-type* parser. The parsed output will dump to the screen, unless one redirects the output to a file.



```
C:\>csp64 -db c:\dump\History > out.csv
```

3.1 Integrated Parsing Algorithms

The ***csp*** tool offers three possible parsing algorithms to choose from; these are detailed below:

1. *Default* option. This option uses the internal SQLite library that is statically linked into the tool to perform a *SQL-Select* statement on the database under analysis. It is sensitive to corrupt databases and will not parse out records from unused or slack space.
2. *Carve* option. (***-carve***). This option uses a TZWorks based set of algorithms to traverse the SQLite data structures to parse the records in the database. It relies on the database's schema and internal tree-based structures to find the data. This option appears to work fine even if the database cannot be opened via the standard SQLite library. When corruption is present, this option will skip bad records and attempts to go to the next one. It also looks at unused space for any records that may be present using the ***-incl_slack*** option.

3. *Signature-base* option. (***-parse_chunk***). This option *does not* make use of the SQLite schema or tree-based structures in the database to locate records. Instead, it looks for certain signatures in order to locate records and parse them. Empirical testing has shown this approach works from either a good database, corrupted database or a partial blob of a database. While this option can pull valid records, it truncates the data when a record spans multiple SQLite-pages. For any records that are truncated, the output will be annotated with a flag identifying it as such.

3.1.1 Algorithms and their Pros/Cons

The benefit of the *default* option is its usefulness for verification and validation purposes. Given that the tool can produce the same output for any of the three available parsing options, one can use the *default* option as the base option to compare other parsing algorithm results. In this way, one can easily verify whether the *carve* option and/or *signature-based* option works, simply by comparing the results to that of the *default SQL-Select* option. More details about this verification are discussed later in a separate section (see section on *Verification and Validation*).

In most cases, the *carve* option (***-carve***) is a better choice over the *default* option, simply because is it returns the same, if not more, results. If invoking the sub-option ***-incl_slack***, the tool has the ability to sense unused space and switches to a *signature-based* scan for those areas.

Surprisingly, the *signature-base* option (***-parse_chunk***) competes very well with the other two options. Keep in mind, this option relies strictly on unique signatures being accurate for its success. While the other two options can dynamically adjust their parsing engine based on the schema identified in the database, the *signature-based* option cannot. Depending on the number of recoverable records in the database, it is possible for signature-based option to extract more records than the other options, however, the user is cautioned, that more records do not necessarily mean accurate data. For example, if one passes in a file that contains the contents of a disk volume, with the intent of extracting all the Chromium artifacts from that image, then the user may get multiple false positives on certain table records. The ***csp*** tool does a good job of statistically pulling out table entries that have many fields versus those tables that only have a few fields. Therefore, certain table entries will have less false positives than others.

The other issue to consider with the *signature-base* option is the merging operation from data in one table to another table (based on some relationship between the tables) may or may not make sense. For example, if a timestamp from one table is merged with data from another table, and the data is not in sync (from a chronological point of view), then the resulting merged record will mislead the investigator of an event's occurrence time-wise. The other pitfall with the *signature-based* scans, which was mentioned earlier, is that approach will truncate the data if a record overflows into multiple databases pages; the *signature-based* scan will only report on data found in the initial page.

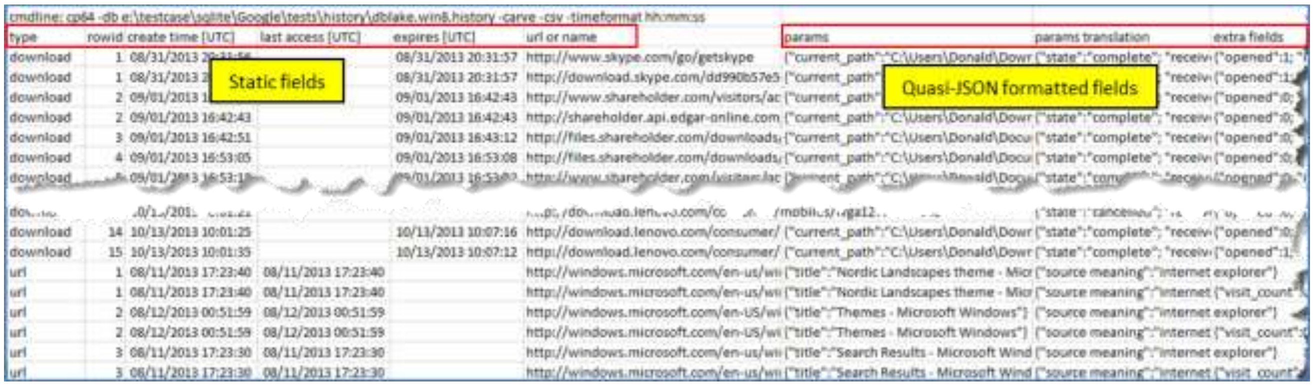
To handle the data accuracy issue, refer to the section on "*Merging of Data between Tables*". In conclusion, despite the negatives for the *signature-based* parse, it is the only choice if analyzing partial chunks of database fragments, whether from memory or disk images.

3.2 Modified CSV Output

When parsing various databases, where a database type can have differing tables and each table translates to differing schemas or fields, one of the challenges in report generation is how can one get all the varying data fields into a common CSV format. The simple answer is to invoke the *Log2Timeline* option (***-csvl2t***), or the *Sleuthkit BodyFile* option (***-bodyfile***). These are excellent options to achieve this, since these formats have custom pre-defined fields. They are defined in such a way, so that the format allows for dissimilar datasets by assuming all record will have at least a timestamp and description of the event that occurred. These formats also contain fields for generic data for notes and comments.

The above formats, because of their nature, can take one record and create multiple CSV entries if an entry contains multiple differing timestamps. Therefore, if one desires to output a single CSV line per record, then one needs to generate their own custom format. Leveraging off of the concept of the ***-csvl2t*** format, one can accomplish this by creating some static fields as well as some general-purpose fields. For the normal ***-csv*** option, the ***csp*** tool does just that. Specifically, there are a few static fields where the types are set, but there are others where a quasi-*JSON* format is used. In this way, many of the fields of a record can be outputted in a way where like-fields, such as *Type of record*, *RowID*, *Timestamp*, and *URL* are static, but the other general-purpose fields can contain differing types of data.

For general-purpose data, the quasi-*JSON* format used by the ***csp*** tool consists of outputting the data in a *name/value* pairing relationship. From the snapshot of the output below, one can see these fields which are identified with the field headers: *params* and *extra fields*. The image below was meant for illustrative purposes, and therefore only shows some of the fields.



cmdline: cp64 -db e:\testcase\sqlite\Google\tests\history\dblake.win8.history -carve -csv -timeformat hh:mm:ss

type	rowid	create time [UTC]	last access [UTC]	expires [UTC]	url or name	params	params translation	extra fields
download	1	08/31/2013 2		08/31/2013 20:31:57	http://www.skype.com/go/getskype	{"current_path":"C:\Users\Donald\Dowr	{"state":"complete"; "receiv	{"opened":1;
download	1	08/31/2013 2		08/31/2013 20:31:57	http://download.skype.com/dd990b57e5	{"current_path"	"receiv	{"opened":1;
download	2	09/01/2013 1		09/01/2013 16:42:43	http://www.shareholder.com/visitors/ac	{"current_path"	"receiv	{"opened":0;
download	2	09/01/2013 16:42:43		09/01/2013 16:42:43	http://shareholder.api.edgar-online.com	{"current_path":"C:\Users\Donald\Dowr	{"state":"complete"; "receiv	{"opened":0;
download	3	09/01/2013 16:42:51		09/01/2013 16:43:12	http://files.shareholder.com/downloads	{"current_path":"C:\Users\Donald\Docu	{"state":"complete"; "receiv	{"opened":0;
download	4	09/01/2013 16:53:05		09/01/2013 16:53:08	http://files.shareholder.com/downloads	{"current_path":"C:\Users\Donald\Docu	{"state":"complete"; "receiv	{"opened":0;
download	[illegible]	09/01/2013 16:53:1		09/01/2013 16:53	http://www.shareholder.com/visitors/ac	{"current_path":"C:\[illegible]\Docu	{"state":"com[illegible]	{"opened":0;
[illegible]		[illegible]			[illegible]	[illegible]	{"state":"cancel[illegible]	[illegible]
download	14	10/13/2013 10:01:25		10/13/2013 10:07:16	http://download.lenovo.com/consumer/	{"current_path":"C:\Users\Donald\Dowr	{"state":"complete"; "receiv	{"opened":0;
download	15	10/13/2013 10:01:35		10/13/2013 10:07:12	http://download.lenovo.com/consumer/	{"current_path":"C:\Users\Donald\Dowr	{"state":"complete"; "receiv	{"opened":1;
url	1	08/11/2013 17:23:40	08/11/2013 17:23:40		http://windows.microsoft.com/en-us/wi	{"title":"Nordic Landscapes theme - Micr	{"source meaning":"internet explorer"}	
url	1	08/11/2013 17:23:40	08/11/2013 17:23:40		http://windows.microsoft.com/en-us/wi	{"title":"Nordic Landscapes theme - Micr	{"source meaning":"internet	{"visit_count"
url	2	08/12/2013 00:51:59	08/12/2013 00:51:59		http://windows.microsoft.com/en-US/wi	{"title":"Themes - Microsoft Windows"}	{"source meaning":"internet explorer"}	
url	2	08/12/2013 00:51:59	08/12/2013 00:51:59		http://windows.microsoft.com/en-US/wi	{"title":"Themes - Microsoft Windows"}	{"source meaning":"internet	{"visit_count":
url	3	08/11/2013 17:23:30	08/11/2013 17:23:30		http://windows.microsoft.com/en-us/wi	{"title":"Search Results - Microsoft Wind	{"source meaning":"internet explorer"}	
url	3	08/11/2013 17:23:30	08/11/2013 17:23:30		http://windows.microsoft.com/en-us/wi	{"title":"Search Results - Microsoft Wind	{"source meaning":"internet	{"visit_count"

3.2.1 Type Designations

In the output there is a column that identifies records by a type. These types are listed below along with where the data comes from.

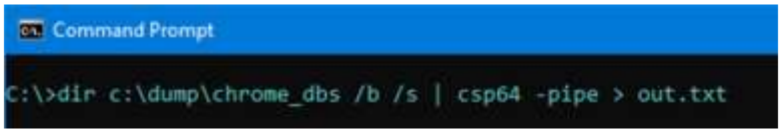
Record Type	Table(s) where the data resides	Database where table(s) reside
Autofill	autofill	Web Data
Cookie	cookie	Cookie
Download	Downloads, download_url_chains	History
Favicon	favicons, icon_mapping, favicon_bitmaps	Favicons
Login	logins	Logins
Shortcut	omni_box_shortcuts	Shortcuts
Thumbnail	thumbnails	Top Sites
Url	urls, visits, keyword_search_terms, visit_source	History
Bounces	Bounces	DIPS
Nel_policies	nel_policies	Reporting and NEL

In addition to the record types shown above, there are some cases were the type is supplemented with some extra words.

- ***Trunc*** - means the data was truncated. This only occurs with using the *signature-based* scan (***-parse_chunk***). This is because the data in the record spans multiple database pages and for *signature-based* scans, only the data in the initial page is parsed.
- ***Blacklisted*** - as applied to the *login* type, and means the *blacklisted_by_user* flag was set in the data. This implies that the login credentials are not stored by Chromium (which may or may not be true).
- ***Created*** - as applied to the *cookie* type, and means the *last access time* equals the *create time.*
- ***Accessed*** - as applied to the *cookie* type, and means the *last access time* does not equals the *create time.*

3.3 Processing Multiple Databases

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



```
c:\>dir c:\dump\chrome_dbs /b /s | csp64 -pipe > out.txt
```

This will process all the databases contained in the *c:\dump\chrome_dbs* folder and subfolders. The results of parsing all the databases found will be put into the file *out.txt*. To help distinguish which lines corresponds 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.

3.4 Merging of Data between Tables

Certain tables contain relationships between them, where data from one table is meant to be combined with another table in order to populate all the fields for a record. The relationships between the Chromium database tables are shown in the section on "*Databases Targeted by this Tool.*" The ***csp*** tool will, by default, try to use these relationships and merge the data between the tables appropriately. Each merged dataset will be treated as a separate record to be outputted into the report. To make this clearer, here's an example: If the records from three tables make two records after the data is merged, only the two merged records will be outputted by this tool in the report.

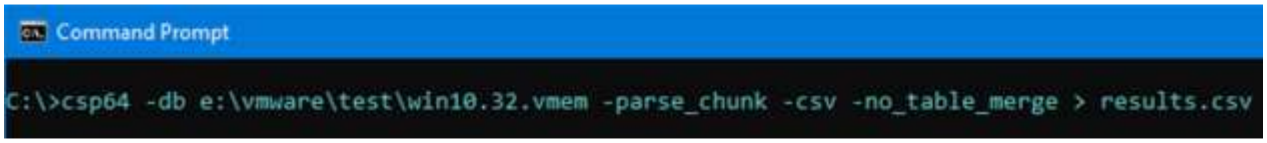
On the flip side, if one has two tables to be merged and they have a '*one to many*' or '*many to one'* relationship, then the tool will try to create a '*one-to-one*' relationship with the results that are outputted. A good example is the *history* database, where a '*one to many*' relationship exists with the *url* table and the *visits* table. One *url* record can have one or more *visits* records. Since the *visits* record has its own timestamp when the visit occurred, to create a proper timeline of events, one needs to duplicate the *url* record data to account for all the *visits* record data. This action of duplication of data from the *url* record creates the '*one-to-one*' relationship in the output. Unfortunately, this gives the perception that there are a large number of duplicate records. Whether it be with the *url* to *visits* relationship or some other table to table relationship, inevitability, there will be duplicates where some of the records outputted will match each other, especially when considering parsing deleted records out of unallocated space. This tool does not make the determination whether the records it parses are duplicated or not; it just outputs all the data.

In some cases, one may not want this merging to take place, and may want to see all the un-merged data from each table separately outputted as a separate record. This behavior can be done by invoking the ***-no_table_merge*** switch. This option only works with the *default* or ***-csv*** output modes (and does not work with ***-csvl2t*** or ***-bodyfile***). This is because not all table records that are parsed by this tool have a timestamp associated with them, which the ***-csvl2t*** and ***-bodyfile*** formats rely on.

The main *use-case* for the ***-no_table_merge***, is when one processes chunk of data (i.e. consider a partial memory dump, volume dump or a partial database file) which contains some Chromium artifacts. In this case, any records extracted from partial tables may relate to one computer's account Chromium data, but not to another account. Alternatively, using the same example, assume there is only one user account on the computer; what could happen is that a parsed timestamp from one table may be out of sequence, from a chronological perspective, from data in another related table. Therefore, any merge operation in the above cases is dubious at best, since there is really no good way to tell if the merge operation will yield accurate results.

3.5 Parsing Chromium Artifacts from Memory or a Disk Image

If one wishes to parse artifacts from a file-based archive that contains a memory or a disk image, then one would use the ***-parse_chunk*** option. During the parsing operation, the tool uses a *signature-based* scan looking for records. Below is an example of performing this operation on a VMWare memory image. Notice we incorporated the ***-no_table_merge*** option as well, since we do not want to merge table data together. This is done as a precaution in case there were multiple instances of Chromium artifacts at one time or another; each instance, in this case, would represent a different user account on the system. Merging table data from one user to another user would yield incorrect and misleading results.


Command Prompt

```
C:\>csp64 -db e:\vmware\test\win10.32.vmem -parse_chunk -csv -no_table_merge > results.csv
```

Notice in the command shown, that we still use the ***-db <file>*** syntax even though the file we are parsing is not a database, but is an image of physical memory stored as a file.

The same type of scan can be done on any image that is not encrypted. The only restriction here is that the image (memory, volume, disk or chunk of data) has to be identical to the system it came from. The key here is the SQLite records being scanned/parsed need to be preserved in their original form.

The last point to mention is if the ***csp*** tool detects that you are passing in a *very large* file for analysis, it will complain if you are not using the option ***-parse_chunk***. This limitation is hardcoded into the tool. Furthermore, it will automatically switch into the mode ***-no_table_merge***. The term '*very large*' in this context are sizes not normal for individual Chromium databases, so an arbitrary size above 130 MB is used for this threshold. The reason for this design decision is to bound what resources the tool can use for parsing; this is a good thing if you want to run it and play well with other applications running on a system. For the tool to do a proper merge operation, all the table records that were parsed either need to be in memory or they need to be in a file base archive (like an SQLite database). In general, when we design our tools, we prefer to maintain a small footprint on the target box and not create additional artifacts on the system under investigation. Therefore, the ***csp*** tool does not explicitly create any temporary files itself; the Operating System, on the other hand, may do this on your behalf, but that is entirely another matter which we cannot control. Taking all these constraints into account, the ***csp*** only operates in memory, and to maintain a small footprint, it flushes any parsed data to the results file so as to reuse the old memory for new records to be parsed.

3.6 Bypassing the Embedded SQLite library

The ***csp*** tool has the SQLite library embedded into the binary. More information about this is discussed in the section *Use of the SQLite Library*. The ***csp*** tool makes use of this library in the default mode when parsing.

Sometimes, however, one may not wish to use the SQLite library for analyzing tables and extracting 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 in 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, the ***-carve*** option. This option will try to traverse the internal SQLite data structures in the database (even corrupted ones), and should extract all the same information as if using the normal SQLite. The difference here is the ***-carve*** option is more immune to database corruption or database lockdown than the *default* option.

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 make sense of any records it finds. The limitations of this option include: (a) it will not be able to handle overflow records between SQLite pages, and (b) it may not be able to provide joins between tables that have a relational aspect. The ***-carve*** option discussed earlier, however, will handle the overflow of data between pages and perform the necessary joins between tables that have dependencies between them. The benefit of the ***-parse_chunk*** option is that it can handle pulling out records from a journal file independently of the main database file, whereas the other 2 options cannot.

3.7 Verification and Validation

All tools need to tested with some form of verification to ensure their results are accurate. Part of that testing is to validate the tool's functionality across different artifact versions. If the tool developer can automate this testing, then it allows the developer to test the tool across many datasets quickly. This in turn quickly identifies inconsistencies and problems so that a wide range of bugs can be diagnosed and fixed.

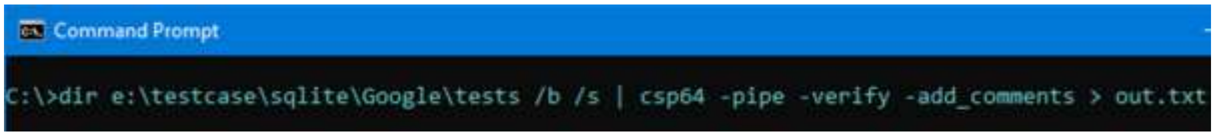
Normally, the developer tries to do as much of this testing before sending a tool out to clients. In the case of Chromium, however, since it has a history of changing the schemas across versions so that they are not backwards compatible, we decided to temporarily add an option for clients to run this type of verification on their own, if they so choose. To this end, the ***csp*** tool incorporates the ***-verify*** option to aid in this purpose.

The ***-verify*** option internally invokes all three parsing engines in sequence to parse the same database so it can compare the results of all three. Simplistically, if all the results match, then the confidence is very high the tool is working as designed. If the results do not match, it will be because a version of Chromium is being analyzed where the tool may work with one of parsing engines, but not the others.

The first parsing engine most likely to have problems will be the *signature-based* parsing, since it more sensitive to schema changes. In contrast, the default *SQL-Select* type parsing engine should be the most robust if there are schema changes, because it will key off of specific field names, which typically are more consistent across versioning. Either way, the purpose of the ***-verify*** option is to provide an internal test to alert a user if any issues are found.

The nice thing about the way this option was implemented, is not only does it check the internal parsers against themselves, but it also outputs critical diagnostic data that can be used by TZWorks to help improve the tool. To ensure no personal information is outputted, the ***-verify*** option sanitizes the results so that it does not contains private/confidential information from the raw artifact. The output primarily contains metadata from the SQLite internal structures. This causes the data generated to be cryptic and only useful for machine type learning/statistics. An additional sub-option was added (***-add comments***) to annotate some additional commentary to the results; this provides some extra information for the user if a test passed or failed and why.

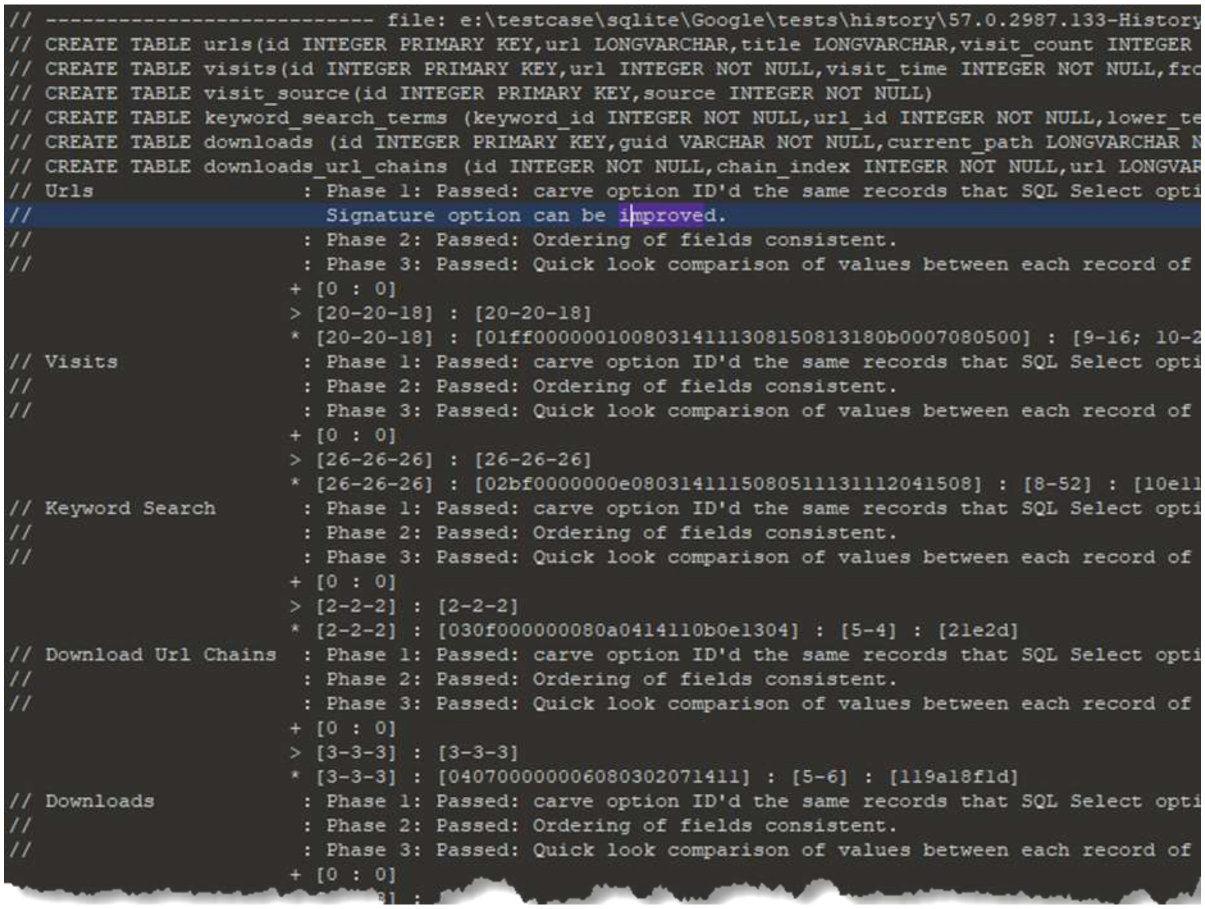
If one wants to test an entire collection of SQLite artifacts from Chromium, one can run this option with the ***-pipe*** switch and point it at the folder/subfolders of Chromium database artifacts. The tool will process all the database/records it finds and incorporate the results into a final report. Below is an example of doing this.



```
C:\>dir e:\testcase\sqlite\Google\tests /b /s | csp64 -pipe -verify -add_comments > out.txt
```

As mentioned earlier, the data produced is mostly cryptic since it contains statistical information about the database and records being parsed. This statistical information, if sent back to TZWorks, will help us improve our parsing engines for future releases.

Below is a screenshot of one of the entries in the results after running this test. For each database processed, there will be information about the various table schemas we are interested in. From this we can see if the schema has been updated from one version to another. In addition, the output shows the number of records parsed by each engine, the signatures found, and so on. Highlighted is a case, where the *signature-based* parsing could be improved along with the data necessary we need to make that improvement.



```
// ---------------------------- file: e:\testcase\sqlite\Google\tests\history\57.0.2987.133-History
// CREATE TABLE urls(id INTEGER PRIMARY KEY,url LONGVARCHAR,title LONGVARCHAR,visit_count INTEGER
// CREATE TABLE visits(id INTEGER PRIMARY KEY,url INTEGER NOT NULL,visit_time INTEGER NOT NULL,fro
// CREATE TABLE visit_source(id INTEGER PRIMARY KEY,source INTEGER NOT NULL)
// CREATE TABLE keyword_search_terms (keyword_id INTEGER NOT NULL,url_id INTEGER NOT NULL,lower_te
// CREATE TABLE downloads (id INTEGER PRIMARY KEY,guid VARCHAR NOT NULL,current_path LONGVARCHAR N
// CREATE TABLE downloads_url_chains (id INTEGER NOT NULL,chain_index INTEGER NOT NULL,url LONGVAR
// Urls                   : Phase 1: Passed: carve option ID'd the same records that SQL Select opti
//                          Signature option can be improved.
//                        : Phase 2: Passed: Ordering of fields consistent.
//                        : Phase 3: Passed: Quick look comparison of values between each record of
                         + [0 : 0]
                         > [20-20-18] : [20-20-18]
                         * [20-20-18] : [01ff000000100803141113081508131 80b0007080500] : [9-16; 10-2
// Visits                 : Phase 1: Passed: carve option ID'd the same records that SQL Select opti
//                        : Phase 2: Passed: Ordering of fields consistent.
//                        : Phase 3: Passed: Quick look comparison of values between each record of
                         + [0 : 0]
                         > [26-26-26] : [26-26-26]
                         * [26-26-26] : [02bf0000000e080314111508051113111 2041508] : [8-52] : [10e11
// Keyword Search         : Phase 1: Passed: carve option ID'd the same records that SQL Select opti
//                        : Phase 2: Passed: Ordering of fields consistent.
//                        : Phase 3: Passed: Quick look comparison of values between each record of
                         + [0 : 0]
                         > [2-2-2] : [2-2-2]
                         * [2-2-2] : [030f000000080a0414110b0e1304] : [5-4] : [21e2d]
// Download Url Chains    : Phase 1: Passed: carve option ID'd the same records that SQL Select opti
//                        : Phase 2: Passed: Ordering of fields consistent.
//                        : Phase 3: Passed: Quick look comparison of values between each record of
                         + [0 : 0]
                         > [3-3-3] : [3-3-3]
                         * [3-3-3] : [0407000000006080302071411] : [5-6] : [119a18f1d]
// Downloads              : Phase 1: Passed: carve option ID'd the same records that SQL Select opti
//                        : Phase 2: Passed: Ordering of fields consistent.
//                        : Phase 3: Passed: Quick look comparison of values between each record of
                         + [0 : 0]
```

One final comment on the ***-verify*** option. This is not a do-everything type built-in test. While it is very capable and provides a wealth of information, the biggest limitation of this test is that it only compares un-merged tables records. Therefore, if there is an error during a merge operation between some *table-to-table* relationship, it is not included in the battery of tests used by the ***-verify*** option. The other testing shortfall is the last (phase 3) test only compare the first two parsing engines resulting values and doesn't consider the third parsing engine (signature-type scan). These shortfalls may be something added in the future, but for now the purpose of this automated testing is to: (a) capture differences in various Chromium formats, (b) identify issues with the various parsing engines in the tool so they can be fixed quickly, and (c) get more empirical results as it pertains to *signature-type* scanning, since this engine at its core relies on statistical data.

4 Use of the SQLite Library

The databases that are targeted by the ***csp*** tool are SQLite databases. For the purposes of the ***csp*** 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 that this is acceptable for this release.

5 Pulling Chromium Artifacts off a Live System

Some of the files used in Chromium are locked down which means other applications cannot read or copy them. To run ***csp***, the files that are processed need to allow the application to at least have read access. If this is an issue, then one needs to look to other tools to copy the artifact data. If you are on a Windows machine, one can use the *TZWorks'* tool ***dup*** (Disk Utility and Packer). It will allow one to copy a file or entire directory even if some of the files are locked down by the operating system. To use ***dup*** to target the Chromium folder, one could use the following command:

dup -copydir C:\Users\<username>\AppData\Local\Google\Chrome\User Data\Default -level 9 -out <results folder>\chrome_data

6 Available Options

Option	Description
-db	Specifies which database file to act on. The format is: ***-db <database or file 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.
-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	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.
-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.
-carve	Experimental option. Bypass the SQLite embedded library and parse using TZWorks internal algorithms. This is useful when the database to be parsed is corrupted and the SQLite library has trouble parsing it.

-incl_slack	Experimental option to look a unused space to see if any records are present. Not required with the ***-parse_chunk*** option. Use this in conjunction with ***-carve*** option to look for delete records.
-parse_chunk	Experimental option. Given a portion (chunk) of the database, this option will examine the data to see if any records exist and parse out the contents. This is a signature-based parse so it can parse out records from chunks of memory or slack space (in the form of a file).
-no_table_merge	This option is for pulling records from an image. It is also used for testing and debugging purposes. If you want to see all the tables that were parsed without merging any relationships, use this option.
-verify	This option is for testing and debugging purposes only. This option runs all 3 parsing engines in the tool (*SQL Select* parse, *Carve* parse and *Signature-based* parse) and reports whether the parsers work at least up to the level of the SQL Select parse. Metadata is generated that can be used to help develop more robust parsing algorithms.
-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.

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. https://cs.chromium.org/chromium/src/docs
2. https://chromium.googlesource.com/chromium/chromium/+/df261d32079bc4e1160c36200657eed26fad5961/content/public/common/page_transition_types_list.h
3. https://developers.google.com/analytics/devguides/collection/analyticsjs/cookie-usage?csw=1
4. Evolution of Chrome Databases, by Ryan Benson, https://dfir.blog/chrome-evolution.
5. Hindsight, by Ryan Benson, https://www.obsidianforensics.com/hindsight
6. SQLite library statically linked into tool [Amalgamation of many separate C source files from SQLite version 3.32.3].
7. SQLite documentation [http://www.sqlite.org].