TZWorks® Safari Artifact Parser (*sap*) Users Guide

Abstract

sap is a standalone, command-line tool that parses artifacts associated with the Safari desktop browser. The tool can target certain SQLite databases, property lists (plists) and cookies that are used in *Safari*. The data can be reported into either CSV or Log2Timeline formats. This tool has working versions for Windows, Linux and OS-X.

*Document applies to v0.19 of **sap***

Updated: Apr 25, 2025

Table of Contents

TZWorks® Safari Artifact Parser (*sap*) Users Guide

Copyright © *TZWorks LLC*
Webpage: http://www.tzworks.com/prototype_page.php?proto_id=51
Contact Information: info@tzworks.com

1 Introduction

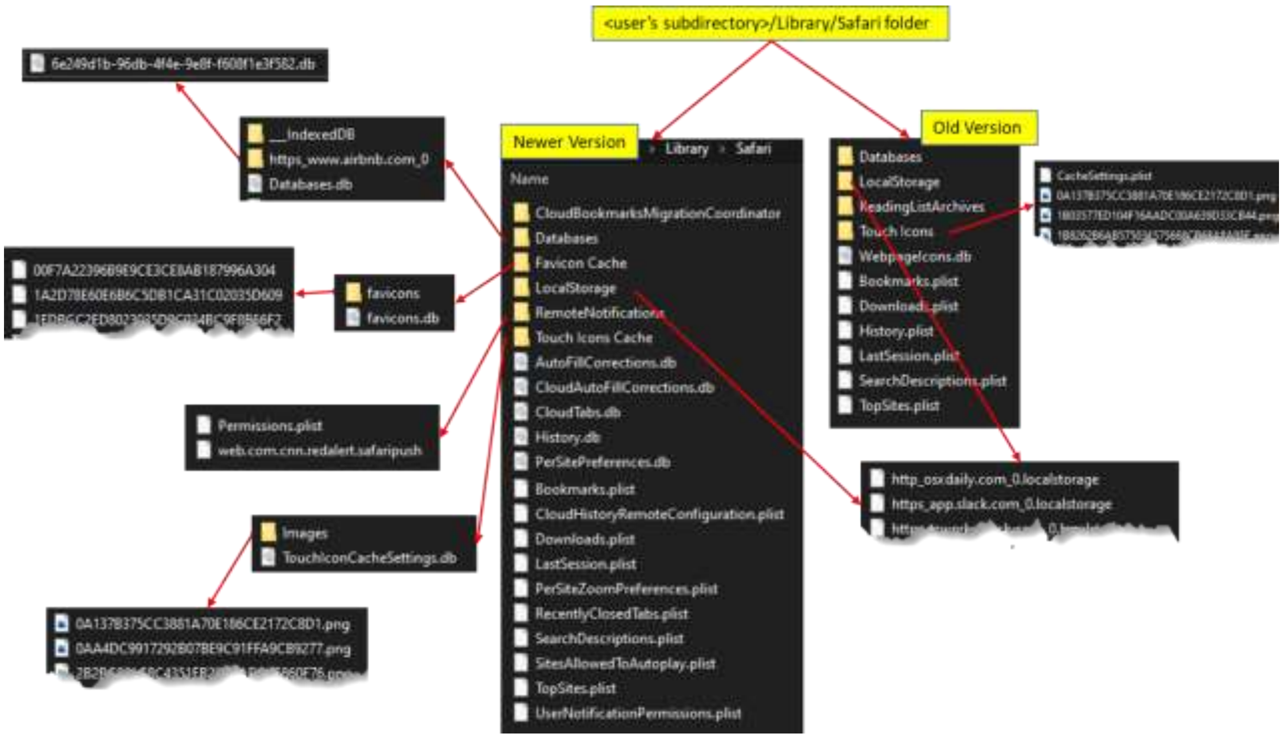
As background, the *WebKit* engine is used in the current *Safari* architecture. Other common browser engines include *Gecko* and *WebKit*. Below is a table to showing where the *Safari* architecture is used, and consequently, which browsers and the respective SQLite tables, the ***sap*** tool targets.

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 last row above, The *Safari* Browser has many artifacts available that the forensics examiner can use in identifying a user's Internet activity. This includes *Safari's* SQLite databases, local storage, associated property lists (*plists*), *cookies* and *cache*. This tool focuses on those artifacts associated with the ***desktop version*** of the browser, however many of these same artifacts appear in the mobile version of the browser as well.

1.1 Location of Safari Artifacts

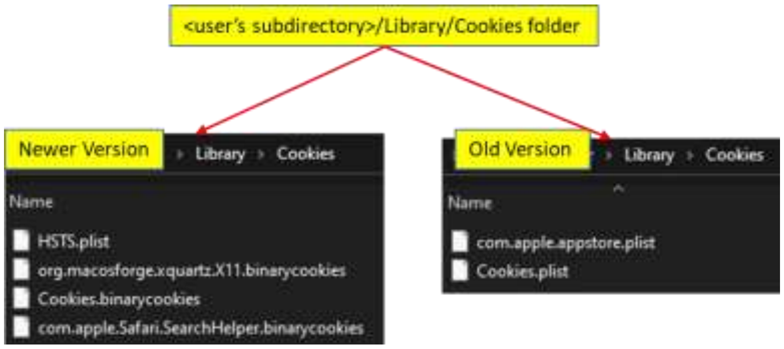
Safari leverages files in the local user's subdirectory; starting with the Library/Safari folder there are various files that are related to *Safari* in some aspect. The locations and diversity types of files are shown in the diagram below.



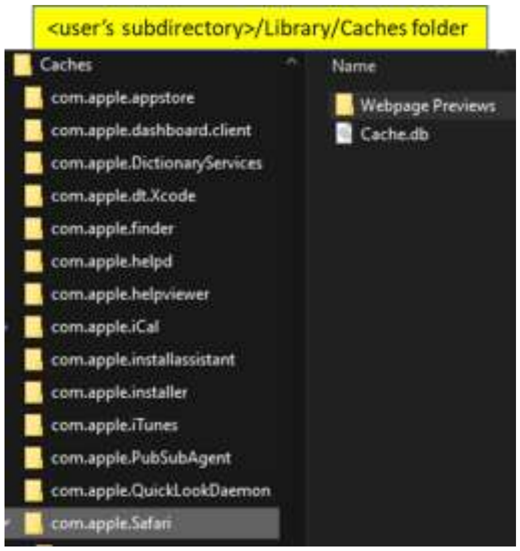
<user's subdirectory>/Library/Safari folder
Newer Version
Library > Safari
Name
CloudBookmarksMigrationCoordinator
Databases
Favicon Cache
LocalStorage
RemoteNotifications
Touch Icons Cache
AutoFillCorrections.db
CloudAutoFillCorrections.db
CloudTabs.db
History.db
PerSitePreferences.db
Bookmarks.plist
CloudHistoryRemoteConfiguration.plist
Downloads.plist
LastSession.plist
PerSiteZoomPreferences.plist
RecentlyClosedTabs.plist
SearchDescriptions.plist
SitesAllowedToAutoplay.plist
TopSites.plist
UserNotificationPermissions.plist
6e249d1b-96db-4f4e-9e8f-f608f1e3f582.db
__IndexedDB
https_www.airbnb.com_0
Databases.db
00F7A22396B9E9CE3CEBAB187996A304
1A2D78E60E6B6C5DB1CA31C02035D609
favicons
favicons.db
Permissions.plist
web.com.cnn.redalert.safaripush
Images
TouchIconCacheSettings.db
0A137B375CC3881A70E186CE2172C8D1.png
0AA4DC9917292B07BE9C91FFA9CB9277.png
Old Version
Databases
LocalStorage
ReadingListArchives
Touch Icons
WebpageIcons.db
Bookmarks.plist
Downloads.plist
History.plist
LastSession.plist
SearchDescriptions.plist
TopSites.plist
CacheSettings.plist
0A137B375CC3881A70E186CE2172C8D1.png
http_osxdaily.com_0.localstorage
https_app.slack.com_0.localstorage

Many of the SQLite files listed are parsed with the ***sap*** tool. However, if a particular file listed is not parsed, it is usually due to a lack of artifact test data (therefore left out of the parser).

When looking at the *cookies* used on a MacOS, there are two types to be aware of. The newer versions consist of a combination of a *HSTS.plist* file along with a number of files that have the *binarycookies* extension. Alternatively, the older version of *Safari* consists of mostly *plist* files. Both of these types are shown in the image below.

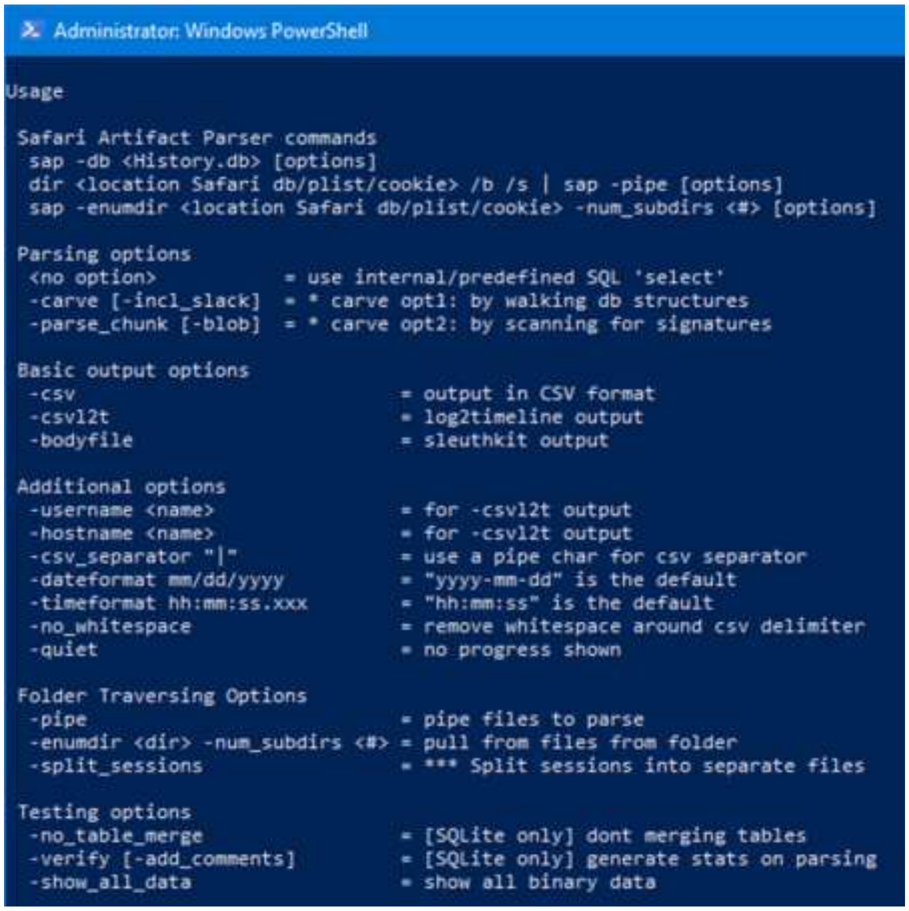


Cached webpages reside in the user's *subdirectory/Library/Cache* folder. For *Safari* related cached webpages, one targets the *com.apple.Safari* subfolder and parses the *Cache.db* file. One point to note - there are many *Cache.db* files located in the Caches subfolder. Given that all the *Cache.db* files appear to all use the same schema, it stands to reason that ***sap*** *should* be able to parse any of them even if they are unrelated to *Safari*. While this may be true to some extent, this tool has not been tested on the non-*Safari* specific ones.


<user's subdirectory>/Library/Caches folder
Caches
com.apple.appstore
com.apple.dashboard.client
com.apple.DictionaryServices
com.apple.dt.Xcode
com.apple.finder
com.apple.helpd
com.apple.helpviewer
com.apple.iCal
com.apple.installassistant
com.apple.installer
com.apple.iTunes
com.apple.PubSubAgent
com.apple.QuickLookDaemon
com.apple.Safari
Name
Webpage Previews
Cache.db

2 How to Use *sap*

The screenshot below shows the options available. The output options are similar to the rest of the TZWorks tools, and can be rendered in one of the three formats: *CSV*, *Log2Timeline*, or *BodyFile* (*Sleuthkit* format).

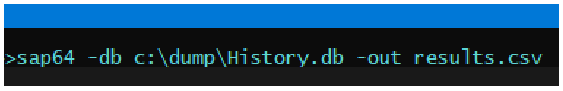


```
Usage

 Safari Artifact Parser commands
  sap -db <History.db> [options]
  dir <location Safari db/plist/cookie> /b /s | sap -pipe [options]
  sap -enumdir <location Safari db/plist/cookie> -num_subdirs <#> [options]

 Parsing options
  <no option>           = use internal/predefined SQL 'select'
  -carve [-incl_slack]  = * carve opt1: by walking db structures
  -parse_chunk [-blob]  = * carve opt2: by scanning for signatures

 Basic output 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.xxx        = "hh:mm:ss" is the default
  -no_whitespace                  = remove whitespace around csv delimiter
  -quiet                          = no progress shown

 Folder Traversing Options
  -pipe                           = pipe files to parse
  -enumdir <dir> -num_subdirs <#> = pull from files from folder
  -split_sessions                 = *** Split sessions into separate files

 Testing options
  -no_table_merge                 = [SQLite only] dont merging tables
  -verify [-add_comments]         = [SQLite only] generate stats on parsing
  -show_all_data                  = show all binary data
```

To process artifact files, ***sap*** can either target a folder, individual *SQLite* database files, individual *plist* files, or *binarycookies* files. The tool will automatically determine the file type version and adjust the parsing engine accordingly. When parsing many subdirectories at once, where each subdirectory is a different account or machine, the tool will dynamically adjust for the type and version of the file so as to preserve the record content when interleaved from one record type to another.

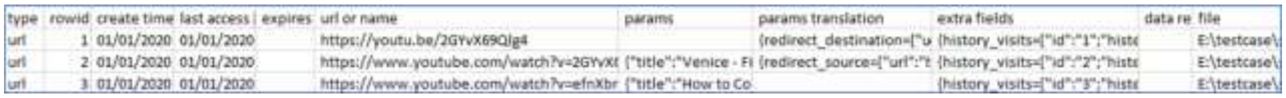
2.1 Targeting Specific files

To target a specific SQLite file, *plist* file, or cookie file, use the ***-db*** option. From ***sap***'s perspective all the file types are treated as if they are a database. If targeting an *SQLite* file, without any specific parsing parameters, the default parser uses the Structured Query Language (SQL) in combination with the statically linked *SQLite* library to extract the records in the various tables in the database. Below is an example of doing this.



```
>sap64 -db c:\dump\History.db -out results.csv
```

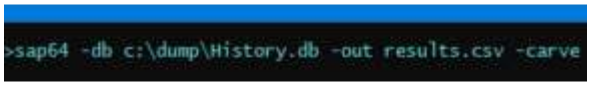
The default output is rendered in pipe delimited text and has a fixed set of fields. These fields are explained in the section on CSV Field Names/Meaning. To allow flexibility rendering differing data types across the tables and databases in the output, some of the fields in the CSV output make use of a quasi-JSON like format; this allows records with different fields across various tables to be rendered in one CSV/delimited format. Below is a sample output.



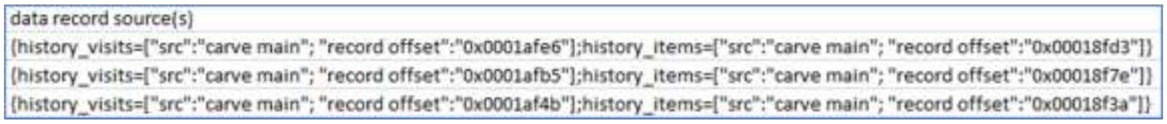
type	rowid	create time	last access	expires	url or name	params	params translation	extra fields	data re	file
url	1	01/01/2020	01/01/2020		https://youtu.be/2GYvX69Qlg4		{redirect_destination=["u	{history_visits=["id":"1";"histe		E:\testcase\
url	2	01/01/2020	01/01/2020		https://www.youtube.com/watch?v=2GYvXt	{"title":"Venice - Fi	{redirect_source=["url":"t	{history_visits=["id":"2";"histe		E:\testcase\
url	3	01/01/2020	01/01/2020		https://www.youtube.com/watch?v=efnXbr	{"title":"How to Co		{history_visits=["id":"3";"histe		E:\testcase\

sap will try to show all the associated fields for each record. Fields that do not have a dedicated column, are shown in the '*extra fields*' column; each field in this column is annotated by a '*name of field/value of data*' pair. Many of the items of interest such as timestamps and URL have their own dedicated columns.

If running ***sap*** with either the ***-carve*** or the ***-parse_chunk*** options, the '*data record sources*' field will be populated with the offset of the record. For example, running the same command above while specifying carve as the parse algorithm yields the same data above, but with the data record sources field populated.



```
>sap64 -db c:\dump\History.db -out results.csv -carve
```



data record source(s)
{history_visits=["src":"carve main"; "record offset":"0x0001afe6"];history_items=["src":"carve main"; "record offset":"0x00018fd3"]}
{history_visits=["src":"carve main"; "record offset":"0x0001afb5"];history_items=["src":"carve main"; "record offset":"0x00018f7e"]}
{history_visits=["src":"carve main"; "record offset":"0x0001af4b"];history_items=["src":"carve main"; "record offset":"0x00018f3a"]}

This gives one the location information necessary to analyze the data in a hex editor to verify the results, should a manual verification of the results be required.

2.2 Integrated Parsing Algorithms (*SQLite*)

sap offers three possible parsing algorithms to choose from when dealing with SQLite databases; these are outlined below:

1. *Default* option. This option uses the internal SQLite library that is statically linked into ***sap*** to perform a *SQL-Select* statement on the database under analysis. It is sensitive to corrupt databases.

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. When corruption is present, this option will skip bad records and will attempt to parse 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 pre-defined signatures in order to locate records and parse them. Empirical testing has shown this approach works from either a fully intact database, a 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.

2.2.1 Algorithms and their Pros/Cons

The benefit of the *default* option is its usefulness for verification and validation purposes. Given that ***sap*** 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.

In most cases, the *carve* option (***-carve***) is a better choice over the *default* option, simply because it returns the same, if not more, results. If invoking the sub-option ***-incl_slack***, the tool has the ability to detect 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 with some exceptions. Keep in mind, the ***-parse_chunk*** option strictly relies on unique signatures being accurate for its success. Also, just because a signature is available, one needs to ensure the signature isn't too simple in the sense it generates many false-positives. 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 *Safari* artifacts from that image, then there may be multiple false positives on certain table records. ***sap*** 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.

2.3 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 rendering all the various data fields into a common CSV format. The simple solution 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 that the format allows for dissimilar datasets by assuming all records will have at least a timestamp and description of the event that occurred. These formats also contain fields for generic data such as 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 some of the fields need to be designated as variable in nature. 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 default or the ***-csv*** option, ***sap*** does just that. Specifically, there are a few static fields where the types are set, but there are others where a *JSON* like format is used. In this way, many of the fields of a record can be outputted in a way where similar 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 *JSON* like format used by ***sap*** consists of outputting the data in a *name/value* pairing relationship.

2.4 Type Designations

For SQLite artifacts, the output will render two types of designations. The first is a result of merging records from tables in accordance with the schema of the database. For this case, the following designations are used:

SQLite Record Type	Table(s) where the data resides	Database where the table(s) reside
favicon	(new) icon_info, page_url, rejected_resources, (older) iconData, IconInfo, PageURL	favicons.db, WebpageIcons.db
url	history_items, history_visits, history_items_to_tags, history_tags	History.db
xxx.localstorage	ItemTable	<url>_0.localstorage

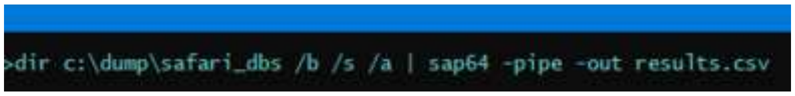
CloudTab	cloud_tab_devices, cloud_tabs	CloudTabs.db
Cache	cfurl_cache_blob_data, cfurl_cache_receiver_data, cfurl_cache_response	Cache.db
Icon Cache	cache_settings	TouchIconCacheSettings.db
<plist type> based on filename	Not applicable	For plist type file
Cookie	Not applicable	For <filename>.binarycookies type files

Alternatively, if merging table records is turned off (via ***-no_table_merge***), then the type designations may specify the actual table name where the data came from. These table names are shown above as well as in the section on "*Databases targeted by **sap***".

In addition to the record types shown above, there are some cases were the type is supplemented with an extra word, such as ***Trunc***, which means the data was truncated.

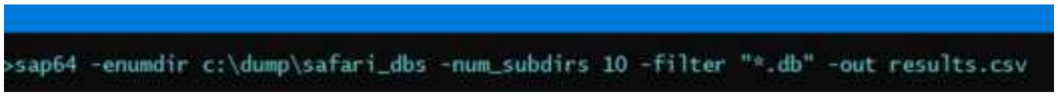
2.5 Processing Multiple Databases

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:



```
>dir c:\dump\safari_dbs /b /s /a | sap64 -pipe -out results.csv
```

To be more discriminating one can use the ***-enumdir*** option along with the sub options ***-num_subdirs*** and ***-filter***. This allows one to target the specified level of subdirectories and files with a certain extension.



```
>sap64 -enumdir c:\dump\safari_dbs -num_subdirs 10 -filter "*.db" -out results.csv
```

The above command will process all databases contained in the *c:\dump\safari_dbs* folder and the 10th level of subfolders. The results of parsing the databases found will be put into the file *results.csv*. To help distinguish which lines corresponds to which database file, an extra field is appended to each record identifying the source database.

2.6 Merging SQLite 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 *Safari* database tables are shown in the section on "Databases Targeted by ***sap***." By default, ***sap*** will try to use these relationships and merge the data between the tables. Each merged dataset will be treated as a separate record to be output into the report. For example, if the records from three tables make two records after the data is merged, only the two merged records will be output by ***sap*** 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 in the results that are output. Unfortunately, this gives the perception that there are a large number of duplicate records. Whether it be with some other table to table relationship, inevitability there will be duplicates where some of the outputted records will match each other. This is especially true when considering parsing deleted records out of unallocated space. ***sap*** 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 output 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 processing chunks of data (i.e. consider a partial memory dump, volume dump or a partial database file) that contain *Safari* artifacts. In this case, any records extracted from partial tables may relate to one user's account *Safari* 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.

2.7 Bypassing the Embedded SQLite library

sap has the SQLite library embedded into the binary. More information about this is discussed in the section *Use of the SQLite Library*. The 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 former, 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 library. The difference here is the ***-carve*** option is more immune to database corruption or database lockdown, than the default option.

The purpose of 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 two options cannot.

2.7.1 Signature parsing logic

When invoking the ***-parse_chunk*** flag, the internal parser resorts to a *signature-based* parsing logic. When considering the number of different Safari SQLite databases, where each database can have one or more tables and each table having a unique schema, the number of signatures the tool can look at can be quite a few. Therefore, to restrict the number of signatures the tool targets the filename of the database that is passed in to determine which signatures to use to find candidate records. This results in two things: (a) it restricts the number of signatures to scan for, and (b) it reduces the number of false positives, in the case where multiple signatures are similar.

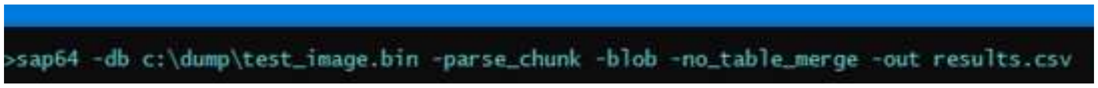
In those situations where the file being passed in was carved from disk or taken from memory, the filename approach described above does not work; specifically, one needs to tell ***sap*** to scan the file for all the signatures internally available. To tell ***sap*** to do this, use the ***-blob*** option in conjunction with ***-parse_chunk***, and the tool will scan through the file looking for various possible records by trying to match a dictionary set of signatures.

2.8 Parsing Safari SQLite Artifacts from Memory or a Disk Image

To parse artifacts from a file-based archive that contains a memory or a disk image, one would use the ***-parse_chunk*** option along with the ***-blob*** option. This tells ***sap*** not to use the filename to determine which *signatures* to use. Instead, the tool will use any *unique signatures* it has to scan for a broad set of record types. The term *unique* is used here to mean the *signature* doesn't produce a large set of false positives.

Even though the tool will allow the user to run it in either the 32-bit or 64-bit version of the binary, one is encouraged to use the 64-bit version. The reason is with a scan that spans a very large disk image, it may be possible to extract a large number of records. The larger the number of records that are extracted, the more memory the tool will consume; using the 64-bit binary can grow the memory needed appropriately, whereas the 32-bit version is limited.

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 are multiple instances of *Safari* 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.



```
>sap64 -db c:\dump\test_image.bin -parse_chunk -blob -no_table_merge -out results.csv
```

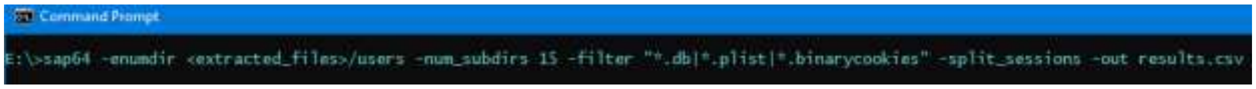
Notice in the command shown, that we still use the ***-db <file>*** syntax even though the file we are parsing is not a database, but an image of physical memory stored as a file.

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 ***sap*** detects a very large file being processed for analysis, it will complain if you are not using the option ***-parse_chunk***. Also, ***sap*** will complain if either the ***-csvl2t*** or ***-bodyfile*** output options are used for large file analysis, since only the ***-csv*** (or the default) output option is allowed for this situation. This limitation is hardcoded into the tool. Furthermore, it will automatically switch into the mode ***-no_table_merge*** for very large files. The term 'very large' in this context are sizes not normal for individual *Safari* databases, so an arbitrary size above 130 MB is used for this threshold.

2.9 Splitting the Safari Sessions into Separate Files

One of the use-cases is to run ***sap*** against a system with multiple accounts, and breakout the parsing results by account into separate files.   To do this, use the option ***-split_sessions***. It can be used with either of the directory enumeration options (***-enumdir*** or ***-pipe***).   The behavior of ***sap*** will take whatever was specified as the output file to be appended with a session number.   For this to work properly, the tool is assuming that the starting folder includes the *user's account folder/subfolders*.   Below is an example using this syntax.



```
E:\>sap64 -enumdir <extracted_files>/users -num_subdirs 15 -filter "*.db|*.plist|*.binarycookies" -split_sessions -out results.csv
```

When the processing is done, a number of files (one per *Safari* session) will be generated. The output notation will the output name specified (in this case "results") prepended with an incremented number along with the folder name used by *Safari* for that session.

3 Databases Targeted by *sap*

sap currently targets the following SQLite databases: (a) *History.db*, (b) *Favicons.db* (or *WebpageIcons.db*), (c) *TouchIconCacheSettings.db*, (d) *CloudTabs.db*, (e) *Cache.db* and (f*) localstorage* related files. This tool focuses on the desktop platform (*MacOS*) of *Safari* artifacts and not the device versions run with *iOS*.

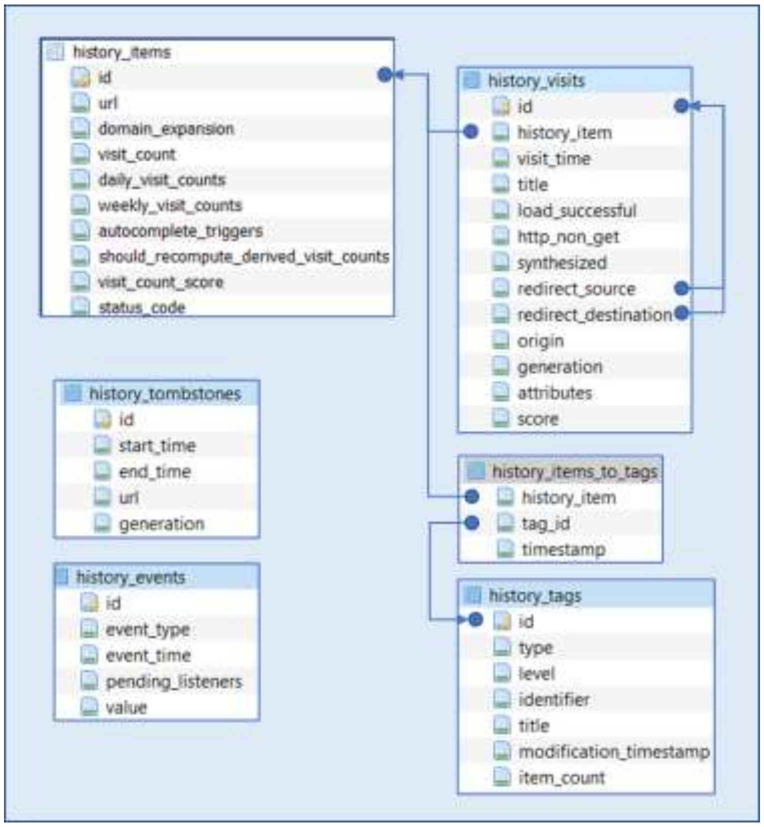
When looking across the various versions of the *Safari* Browser over time, the schemas of the databases have changed somewhat. The database schema can be thought of as the roadmap that defines the fields and the type of data in each field that comprise a record in the table (where one or more tables reside in a database). More often, however, the older browser versions made more extensive use of *plist* (property list) files; the newer browser versions have evolved some of the *plist* files into SQLite files.

The change in schemas across different browser versions as well as the use of *plists* in some of the older versions is something that needed to be taken into account when designing ***sap***. The design allows the tool to dynamically detect and adjust to varying schemas and/or *plist* usage as they are encountered during the processing.

In addition to the auto-schema detection, ***sap*** allows the user to parse a target SQLite database in three ways. (1) The first way makes use of the standard SQL (Structure Query Language) to parse the records. The SQL syntax is internal to the tool, so the user is not required to have any knowledge about SQL or its syntax. For this option to be available, the SQLite library was statically linked into ***sap***, which eliminates the need for the SQLite dynamic library to be present to run the tool. (2) The second approach allows the user to instruct the tool to parse each record by traversing the internal SQLite structures as they are encountered. This option does not use any part of the standard SQLite library, but utilizes the TZWorks' internally designed libraries. The library allows ***sap*** to extract records from a corrupted database and annotate the exact offset of the data where it was found. This enables one to easily validate it later with a hex-editor. (3) The third, and final approach, uses a signature-based parse. While this option is more limited in merging records from one table to another, this turns out to be a unique way in parsing a blob of data whether it be from memory or from a fragment of a database. All three approaches are designed into ***sap*** for the analyst to use.

3.1 *History.db* Database

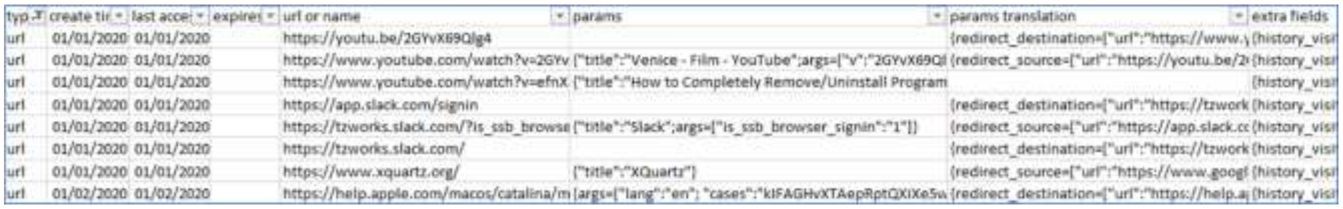
Safari's History.db database located in the *<user's subdir>/Library/Safari/* folder has a number of tables of interest to the analyst. Below is a diagram of these tables and their relationships to each other. Keep in mind not all the tables, as well as fields in the tables, may be present in the older *Safari* browser versions. The same can be said of the fields that comprise each of the tables.



The table relationships are shown by the lines connecting one table to another. These relationships will have an effect on the number of records that will be outputted by ***sap***. For example, the tables *history_items* and *history_visits* have what is called a 'one to many' relationship. The *history_visits* may have many linked records to only one entry in the *history_items* table. Therefore, after merging the data from the *history_visits* table to the *history_items* table, one most likely will get more records in the output of the report then the number of records in the *history_items* 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 *history_items* entry and one *history_visits* entry. If there was a second *history_visits* entry for the same *history_items* entry, that would constitute a separate output line. Outputting the data this way allows the various timestamps recorded to be digested better by other tools.

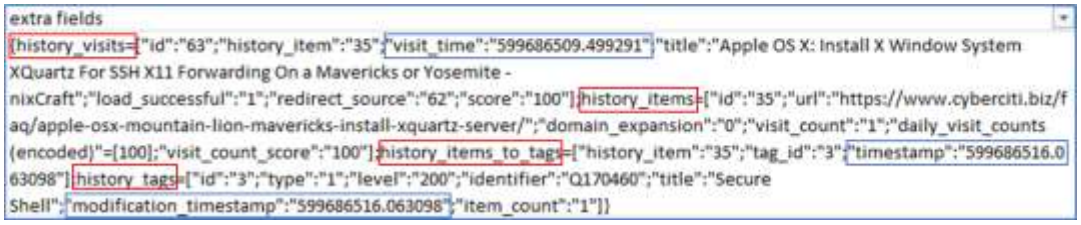
This behavior exists across other tables as well, assuming there are multiple entries from one table referencing a single entry in another table.

Below is a sample output from the fields in the *History.db* database. One should note, while the timestamps shown are for *create* and *last access*, the timestamps can come from a number of sources with this database. (a) *history_visits::visit_time*, (b) *history_tags::modification_timestamp* or (c) *history_items_to_tags::timestamp*. The *visit_time* is used for the create time and the others (if available) are used for the last access time.



typ	create tir	last acce:	expire:	url or name	params	params translation	extra fields
url	01/01/2020	01/01/2020		https://youtu.be/2GYvX69Qlg4		{redirect_destination=["url":"https://www.\	{history_visi
url	01/01/2020	01/01/2020		https://www.youtube.com/watch?v=2GYv	["title":"Venice - Film - YouTube";args=["v":"2GYvX69Ql	{redirect_source=["url":"https://youtu.be/2	{history_visi
url	01/01/2020	01/01/2020		https://www.youtube.com/watch?v=efnX	["title":"How to Completely Remove/Uninstall Program		{history_visi
url	01/01/2020	01/01/2020		https://app.slack.com/signin		{redirect_destination=["url":"https://tzwork	{history_visi
url	01/01/2020	01/01/2020		https://tzworks.slack.com/?is_ssb_browse	["title":"Slack";args=["is_ssb_browser_signin":"1"]]	{redirect_source=["url":"https://app.slack.cc	{history_visi
url	01/01/2020	01/01/2020		https://tzworks.slack.com/		{redirect_destination=["url":"https://tzwork	{history_visi
url	01/01/2020	01/01/2020		https://www.xquartz.org/	["title":"XQuartz"]	{redirect_source=["url":"https://www.googl	{history_visi
url	01/02/2020	01/02/2020		https://help.apple.com/macos/catalina/m	{args=["lang":"en"; "cases":"kIFAGHvXTAepRptQXIXe5w	{redirect_destination=["url":"https://help.a	{history_visi

If desiring to know which timestamp is represented, one can examine the '*extra fields*' which has the raw data for each of the fields. An example of an expanded '*extra fields*' is shown in the screenshot below.

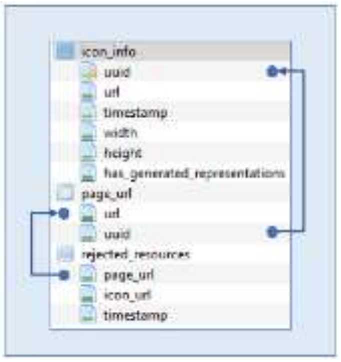


```
extra fields
{history_visits=["id":"63";"history_item":"35";"visit_time":"599686509.499291";"title":"Apple OS X: Install X Window System
XQuartz For SSH X11 Forwarding On a Mavericks or Yosemite -
nixCraft";"load_successful":"1";"redirect_source":"62";"score":"100"];history_items=["id":"35";"url":"https://www.cyberciti.biz/f
aq/apple-osx-mountain-lion-mavericks-install-xquartz-server/";"domain_expansion":"0";"visit_count":"1";"daily_visit_counts
(encoded)"=[100];"visit_count_score":"100"];history_items_to_tags=["history_item":"35";"tag_id":"3";"timestamp":"599686516.0
63098"];history_tags=["id":"3";"type":"1";"level":"200";"identifier":"Q170460";"title":"Secure
Shell";"modification_timestamp":"599686516.063098";"item_count":"1"]}
```

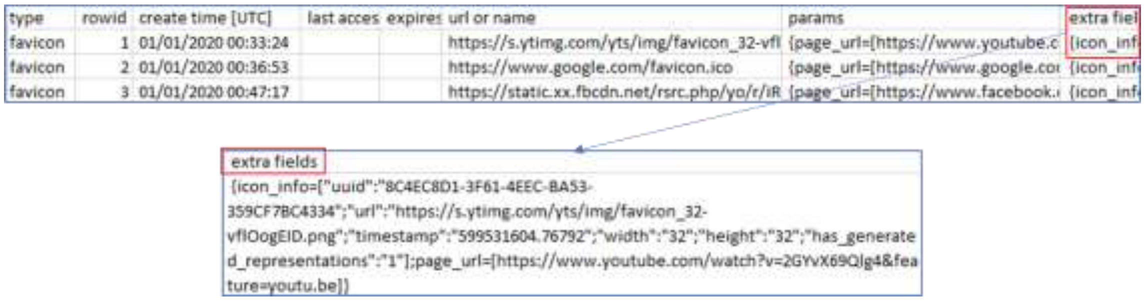
3.2 *Favicons.db* or *WebpageIcons.db* Database

Depending on the *Safari* version, there will be one of two website icon datatypes. The primary data of interest are the timestamps and the URL that was visited.

The *Favicons.db* database is found in the newer version of the browser and is located in the *<user's subdir>/Library/Safari/Favicon Cache/* folder. It has 3 tables of interest to the analyst. Below is a diagram of these tables and their relationships to each other.



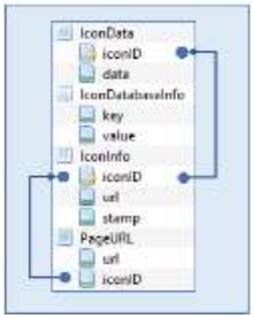
Below is a sample output from the fields in the *Favicons.db* database.



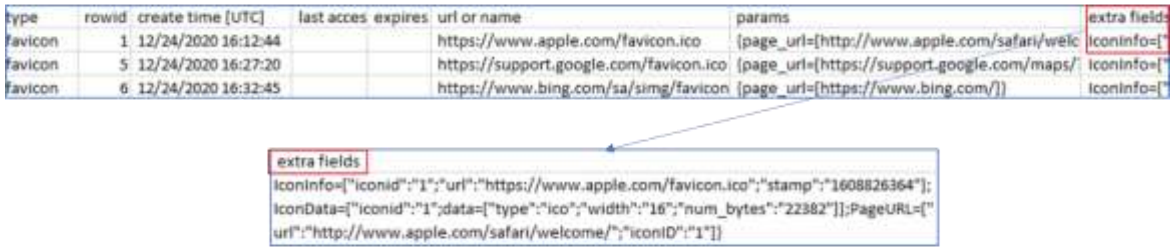
type	rowid	create time [UTC]	last acces	expires	url or name	params	extra fiel
favicon	1	01/01/2020 00:33:24			https://s.ytimg.com/yts/img/favicon_32-vfl	{page_url=[https://www.youtube.c	{icon_inf
favicon	2	01/01/2020 00:36:53			https://www.google.com/favicon.ico	{page_url=[https://www.google.coı	{icon_inf
favicon	3	01/01/2020 00:47:17			https://static.xx.fbcdn.net/rsrc.php/yo/r/iR	{page_url=[https://www.facebook.ı	{icon_inf

extra fields
{icon_info=["uuid":"8C4EC8D1-3F61-4EEC-BA53-359CF7BC4334";"url":"https://s.ytimg.com/yts/img/favicon_32-vflOogEID.png";"timestamp":"599531604.76792";"width":"32";"height":"32";"has_generated_representations":"1"];page_url=[https://www.youtube.com/watch?v=2GYvX69Qlg4&feature=youtu.be]}

The older version of the website icon database is the *WebpageIcons.db*. This is located in the *<user's subdir>/Library/Safari/* folder. The tables of interest as well as their relationships are shown below.



The sample output is shown below. As one can see, the main fields are similar from the new browser version; the only real changes between the versions are in the data reflected in the '*extra_fields*' column.

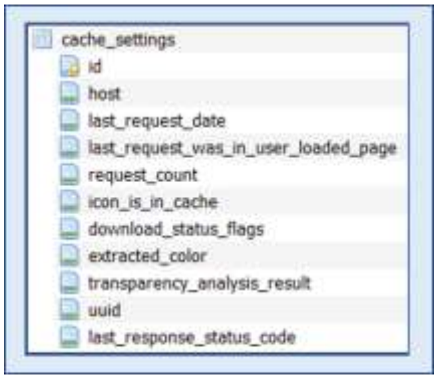


type	rowid	create time [UTC]	last acces	expires	url or name	params	extra fields
favicon	1	12/24/2020 16:12:44			https://www.apple.com/favicon.ico	{page_url=[http://www.apple.com/safari/welc	IconInfo=["
favicon	5	12/24/2020 16:27:20			https://support.google.com/favicon.ico	{page_url=[https://support.google.com/maps/'	IconInfo=["
favicon	6	12/24/2020 16:32:45			https://www.bing.com/sa/simg/favicon	{page_url=[https://www.bing.com/]}	IconInfo=["

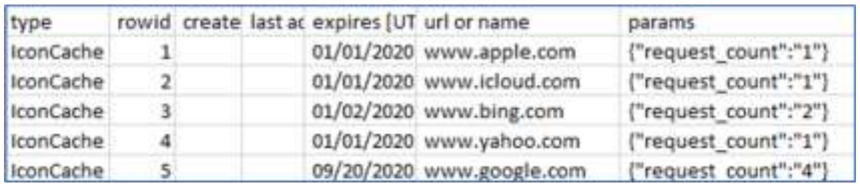
extra fields
IconInfo=["iconid":"1";"url":"https://www.apple.com/favicon.ico";"stamp":"1608826364"];IconData=["iconid":"1";data=["type":"ico";"width":"16";"num_bytes":"22382"]];PageURL=["url":"http://www.apple.com/safari/welcome/";"iconID":"1"]}

3.3 *TouchIconCacheSettings.db* Database

The *TouchIconCacheSettings.db* is located in the *<user's subdir>/Library/Safari/Touch Icon Cache/* folder. Fields of interest are the *host* that was visited, the *last request* date, and the *request count*. One can also pull the images from the next level subdirectory labeled *Images*. Below are the fields of the *cache_settings* table.


cache_settings
id
host
last_request_date
last_request_was_in_user_loaded_page
request_count
icon_is_in_cache
download_status_flags
extracted_color
transparency_analysis_result
uuid
last_response_status_code

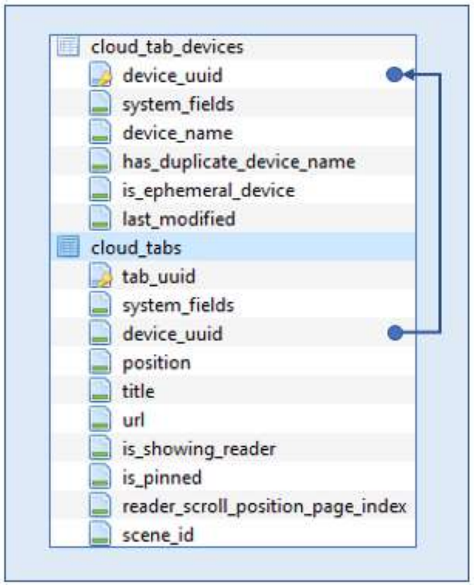
Below is a sample output from the fields in the *TouchIconCacheSettings.db* database.



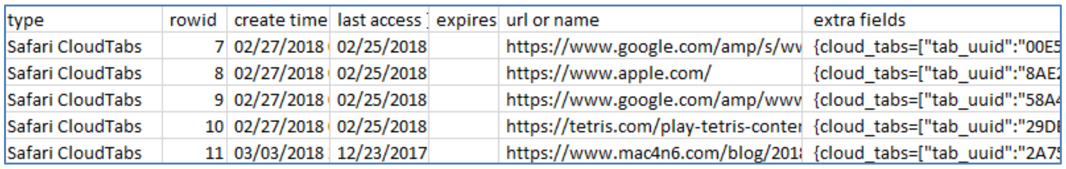
type	rowid	create	last ac	expires (UT	url or name	params
IconCache	1			01/01/2020	www.apple.com	{"request_count":"1"}
IconCache	2			01/01/2020	www.icloud.com	{"request_count":"1"}
IconCache	3			01/02/2020	www.bing.com	{"request_count":"2"}
IconCache	4			01/01/2020	www.yahoo.com	{"request_count":"1"}
IconCache	5			09/20/2020	www.google.com	{"request_count":"4"}

3.4 *CloudTabs.db* Database

The *CloudTabs.db* is located in the *<user's subdir>/Library/Safari/* folder. This also documents URL and timestamps related to a device. The tables shown below are connected via the device's universal unique identifier and exhibits a *one-to-many* relationship; there can be one or more *cloud_tabs* entries for a single *cloud_tab_device* entry. The fields for these tables are shown below:


cloud_tab_devices
device_uuid
system_fields
device_name
has_duplicate_device_name
is_ephemeral_device
last_modified
cloud_tabs
tab_uuid
system_fields
device_uuid
position
title
url
is_showing_reader
is_pinned
reader_scroll_position_page_index
scene_id

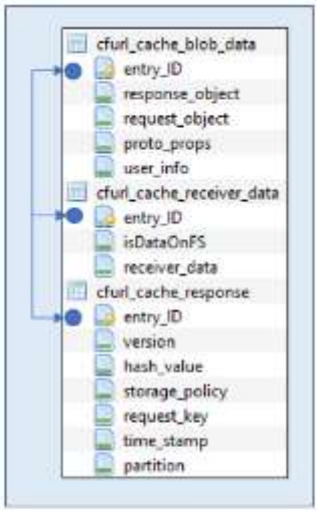
Below is a sample output from the fields in the *CloudTabs.db* database.



type	rowid	create time	last access	expires	url or name	extra fields
Safari CloudTabs	7	02/27/2018	02/25/2018		https://www.google.com/amp/s/wv	{cloud_tabs=["tab_uuid":"00E5
Safari CloudTabs	8	02/27/2018	02/25/2018		https://www.apple.com/	{cloud_tabs=["tab_uuid":"8AE2
Safari CloudTabs	9	02/27/2018	02/25/2018		https://www.google.com/amp/wwv	{cloud_tabs=["tab_uuid":"58A4
Safari CloudTabs	10	02/27/2018	02/25/2018		https://tetris.com/play-tetris-conter	{cloud_tabs=["tab_uuid":"29DI
Safari CloudTabs	11	03/03/2018	12/23/2017		https://www.mac4n6.com/blog/201:	{cloud_tabs=["tab_uuid":"2A75

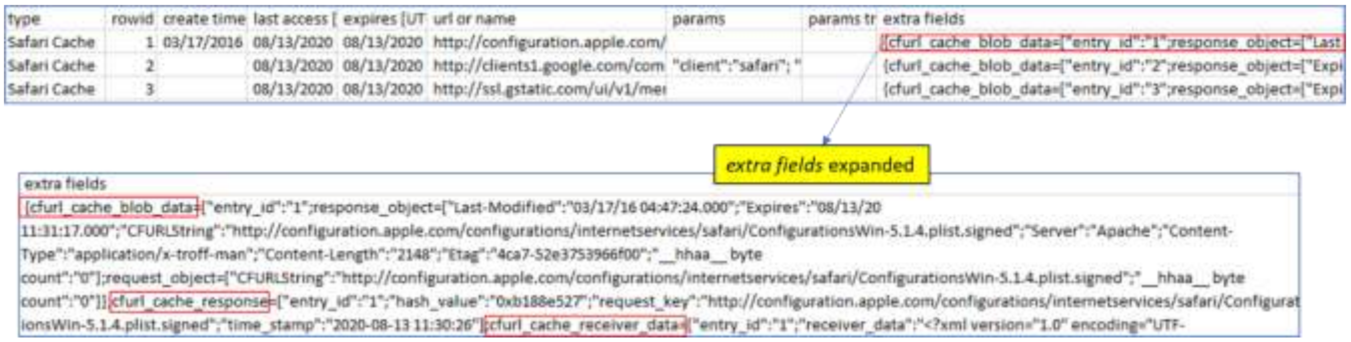
3.5 *Cache.db* Database

A number of *Cache.db* exist. All of them are located in the *<user's subdir>/Library/Caches* folder. The one specific for *Safari* is located in a lower subdirectory, at *<user's subdir>/Library/Caches/com.apple.Safari/* or for newer versions of the OS it is located in the subdirectory *<user's subdir>/Library/Containers/Safari/Data/Library/Caches/com.apple.Safari/*. All the *Cache.db* in the various subfolders have the same schema and therefore could be parsed by this tool. However, ***sap*** was only tested against the *Safari* specific *Cache.db*. The tables and their respective fields used (for schema version 202) are shown below.



The *Cache.db* records both the request from the browser and the response from the server handling the requested webpage. These fields are the *request_obj*ect and the *response object*, respectively. ***sap*** parses out this data and renders the data to the analyst, including: *URL*, timestamps and type of data that was transmitted/received.

Below is a sample output from the fields in this database. Many of the other parameters not shown in their dedicated column but are collected into the '*extra fields*' column. This allows the analyst to review any details that may become pertinent if a finer examination is required. Due to the large amount of data packed into the '*extra fields*' column, the output may have problems rendering in Excel or another spreadsheet tool.



type	rowid	create time	last access [	expires [UT	url or name	params	params tr	extra fields
Safari Cache	1	03/17/2016	08/13/2020	08/13/2020	http://configuration.apple.com/			[cfurl_cache_blob_data=["entry_id":"1";response_object=["Last
Safari Cache	2		08/13/2020	08/13/2020	http://clients1.google.com/com	"client":"safari"; "		[cfurl_cache_blob_data=["entry_id":"2";response_object=["Expi
Safari Cache	3		08/13/2020	08/13/2020	http://ssl.gstatic.com/ui/v1/mei			[cfurl_cache_blob_data=["entry_id":"3";response_object=["Expi

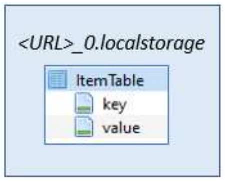
```
extra fields
[cfurl_cache_blob_data=["entry_id":"1";response_object=["Last-Modified":"03/17/16 04:47:24.000";"Expires":"08/13/20
11:31:17.000";"CFURLString":"http://configuration.apple.com/configurations/internetservices/safari/ConfigurationsWin-5.1.4.plist.signed";"Server":"Apache";"Content-
Type":"application/x-troff-man";"Content-Length":"2148";"Etag":"4ca7-52e3753966f00";"__hhaa__byte
count":"0"];request_object=["CFURLString":"http://configuration.apple.com/configurations/internetservices/safari/ConfigurationsWin-5.1.4.plist.signed";"__hhaa__byte
count":"0"]][cfurl_cache_response=["entry_id":"1";"hash_value":"0xb188e527";"request_key":"http://configuration.apple.com/configurations/internetservices/safari/Configurat
ionsWin-5.1.4.plist.signed";"time_stamp":"2020-08-13 11:30:26"][cfurl_cache_receiver_data=["entry_id":"1";"receiver_data":"<?xml version="1.0" encoding="UTF-
```

One thing to note about this artifact.   It goes without saying the above parsing assumes the tool can extract much of the field data as possible. Since this particular artifact is larger in size, when compared to some of the other artifacts, many of the records usually span more than one *SQLite* page.   While this is not a problem for the ***-carve*** or *default* options, this does present more challenges for ***-parse_chunk*** option. Why? Per the discussion previously… "*The pitfall with the signature-based scan [eg. -parse_chunk] … is [it] 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*".

This is something to keep in mind when looking to parse *Cache.db* files.

3.6 *<url>_0.localstorage* related Databases

There are a number of SQLite databases associated with the *LocalStorage*. These databases are located in the *<user's subdir>/Library/Safari/LocalStorage/* folder. Each database in this folder will be named with the URL that was visited with an index number and the *localstorage* extension. These databases contain various key/value pairs that relate to the metadata of the cookies of that webpage.   The schema of the database is very simple in that it only contains a key and a value as shown below. The value field, however, can contained a number of nested separate key/value pairs in a JSON format.



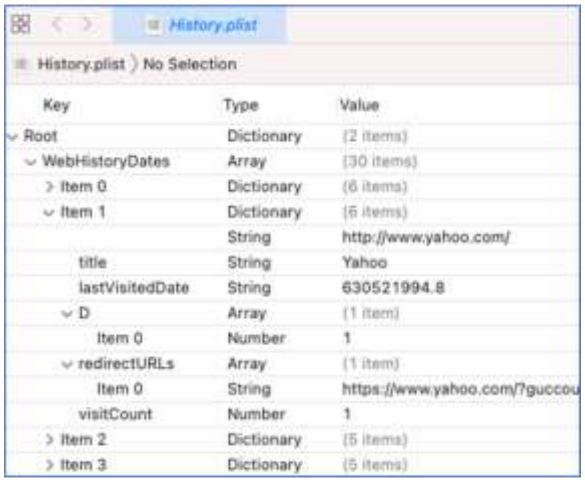
4 Property Lists Targeted by this tool

Property Lists (or *plists*) are used throughout all the applications that run on Apple products. They are used for all sorts of things, like configuration data about bundles/applications, user's settings, and logging state information.   There are various formats used in property lists, the two formats that are most prevalent are either: XML, which is text, or binary which is much more efficient from a storage standpoint. The XML format is able to be read by any text viewer, however, the binary version requires a parser to translate the data packing into something human readable, whether that be in XML, JSON, or some other viewable format.

Originally the intent of ***sap*** was to target SQLite database files, but after researching the fields in various *Safari* SQLite databases, it became apparent that certain fields in these SQLite structures contained blobs of data. Some of these blobs when looked at with a hex-editor, were embedded *plists*. Therefore, adding a *plist* parser to this tool was essential. After the decision was made to add *plist* parsing, the next logical step was to modify the tool to parse both embedded and file *plist* data. The screenshots of the *plist* structures shown in the next subsections were generated from *Apples' XCode* tool. It is very useful in viewing and/or editing *plist* files.

4.1 *History.plist*

This *plist* is used in older *Safari* browsers and acts like a database of visited websites. The *History.plist* is located in the *<user's subdir>/Library/Safari/* folder. It contains the *URL* of the website that was visited, the number of times the website was viewed, the *last visited* timestamp, and which website it was redirected from. The *History.plist* file was later replaced by the SQLite *History.db* database file with the new version of the browsers. To ensure that both the older and newer versions of *Safari* could be handled, ***sap*** can parse either one if it is passed as a source database. Below is the structure of this *plist* file.

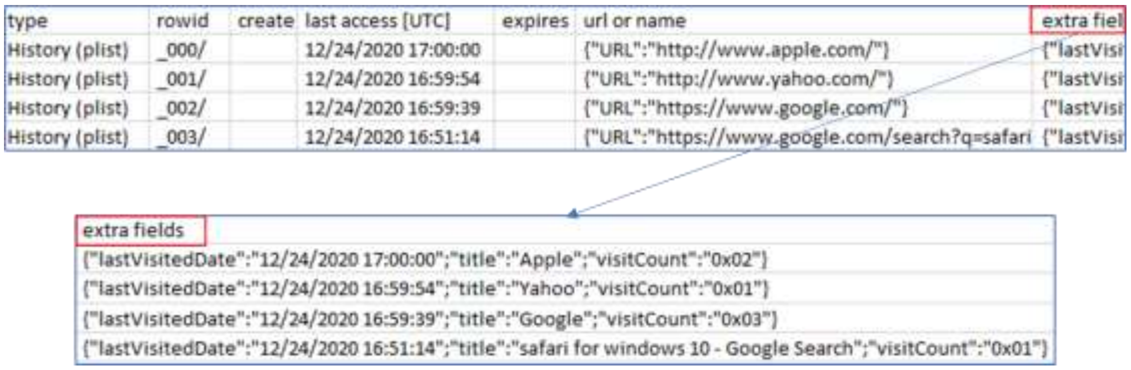


Key	Type	Value
Root	Dictionary	(2 items)
WebHistoryDates	Array	(30 items)
Item 0	Dictionary	(6 items)
Item 1	Dictionary	(6 items)
	String	http://www.yahoo.com/
title	String	Yahoo
lastVisitedDate	String	630521994.8
D	Array	(1 item)
Item 0	Number	1
redirectURLs	Array	(1 item)
Item 0	String	https://www.yahoo.com/?guccou
visitCount	Number	1
Item 2	Dictionary	(5 items)
Item 3	Dictionary	(5 items)

Above is a sample output from the fields in the *History.plist* database. For the *plist* output, ***sap*** uses the same field headers that were used for the SQLite output. In this way, the SQLite data and the *plist* data can co-exist in the same output.

For this case, the content of fields in the *History.plist* are not as extensive as those in the SQLite counterpart (*History.db*), however much of the pertinent data is still there. Since '*rowid*' for *plists* does not apply (since this field was originally geared toward SQLite data), it is used to store the *subpath* of the *plist* entry. For an array datatype, an index is appended to the *subpath*.

For those fields in the *plist* that don't have a designated field name, they are lumped into the '*extra fields*' column. In this way, all the data can be represented for each record.



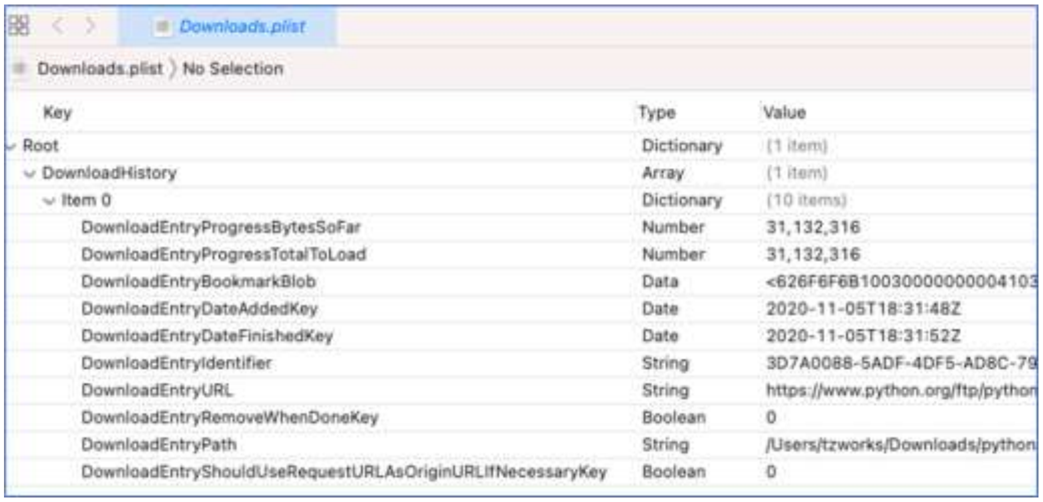
type	rowid	create	last access [UTC]	expires	url or name	extra fiel
History (plist)	_000/		12/24/2020 17:00:00		{"URL":"http://www.apple.com/"}	{"lastVisi
History (plist)	_001/		12/24/2020 16:59:54		{"URL":"http://www.yahoo.com/"}	{"lastVisi
History (plist)	_002/		12/24/2020 16:59:39		{"URL":"https://www.google.com/"}	{"lastVisi
History (plist)	_003/		12/24/2020 16:51:14		{"URL":"https://www.google.com/search?q=safari	{"lastVisi

extra fields
{"lastVisitedDate":"12/24/2020 17:00:00";"title":"Apple";"visitCount":"0x02"}
{"lastVisitedDate":"12/24/2020 16:59:54";"title":"Yahoo";"visitCount":"0x01"}
{"lastVisitedDate":"12/24/2020 16:59:39";"title":"Google";"visitCount":"0x03"}
{"lastVisitedDate":"12/24/2020 16:51:14";"title":"safari for windows 10 - Google Search";"visitCount":"0x01"}

4.2 *Downloads.plist*

This *plist* is used in both the older and new *Safari* browsers and acts like a database of downloaded files. Even though it is available with the newer browser, it appears from empirical data that it does not account for every download. This database contains the (a) *URL* of the file that was downloaded, (b) time it started, (c) time it finished, (d) number of bytes it took and (e) where the file was stored.

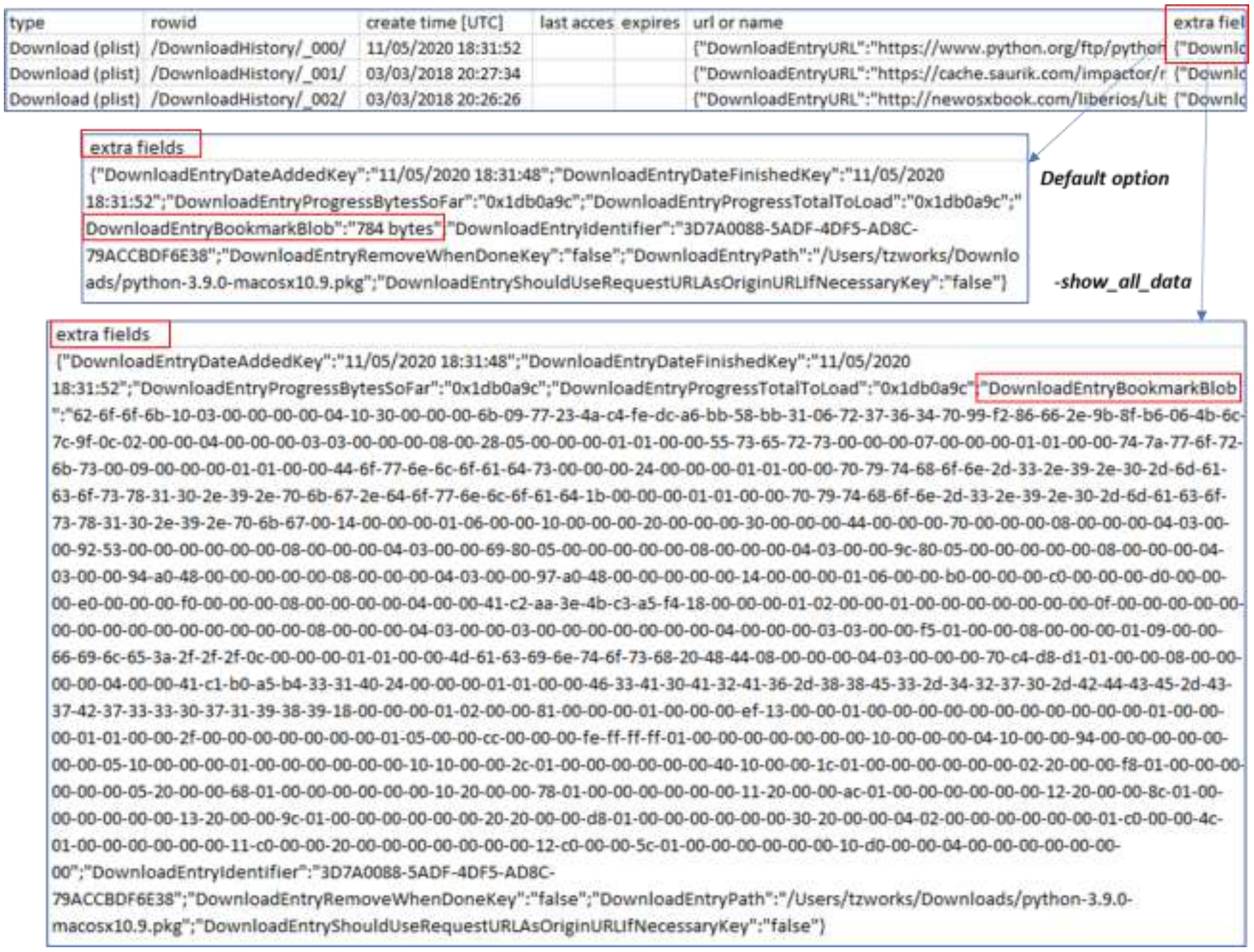
The *Downloads.plist* file is located in the *<user's subdir>/Library/Safari/* folder. Below is the structure of this *plist* file.



Key	Type	Value
Root	Dictionary	(1 item)
DownloadHistory	Array	(1 item)
Item 0	Dictionary	(10 items)
DownloadEntryProgressBytesSoFar	Number	31,132,316
DownloadEntryProgressTotalToLoad	Number	31,132,316
DownloadEntryBookmarkBlob	Data	<626F6F6B100300000000004103
DownloadEntryDateAddedKey	Date	2020-11-05T18:31:48Z
DownloadEntryDateFinishedKey	Date	2020-11-05T18:31:52Z
DownloadEntryIdentifier	String	3D7A0088-5ADF-4DF5-AD8C-79
DownloadEntryURL	String	https://www.python.org/ftp/python
DownloadEntryRemoveWhenDoneKey	Boolean	0
DownloadEntryPath	String	/Users/tzworks/Downloads/python
DownloadEntryShouldUseRequestURLAsOriginURLIfNecessaryKey	Boolean	0

Below is a sample output from the fields in the *Downloads.plist* database.   Similar to the *History plist* output, the '*extra fields*' column has the complete set of data for the record. For binary data, the value of the key/value pair is listed as the number of bytes as shown in for the key name

"*DownloadEntryBookmarkBlob*". To see all the raw bytes, use the option ***-show_all_data***. This will output the raw data shown in the third screen shot. The date shown for the 'create time' is the time value associated with "*DownloadEntryDateFinishedKey*".



type	rowid	create time [UTC]	last acces	expires	url or name	extra fiel
Download (plist)	/DownloadHistory/_000/	11/05/2020 18:31:52			{"DownloadEntryURL":"https://www.python.org/ftp/pythor	{"Downlc
Download (plist)	/DownloadHistory/_001/	03/03/2018 20:27:34			{"DownloadEntryURL":"https://cache.saurik.com/impactor/r	{"Downlc
Download (plist)	/DownloadHistory/_002/	03/03/2018 20:26:26			{"DownloadEntryURL":"http://newosxbook.com/liberios/Lit	{"Downlc

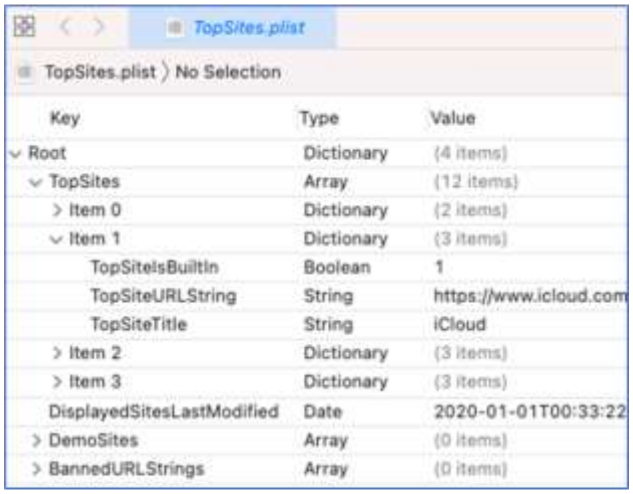
```
extra fields
{"DownloadEntryDateAddedKey":"11/05/2020 18:31:48";"DownloadEntryDateFinishedKey":"11/05/2020
18:31:52";"DownloadEntryProgressBytesSoFar":"0x1db0a9c";"DownloadEntryProgressTotalToLoad":"0x1db0a9c";"
DownloadEntryBookmarkBlob":"784 bytes";"DownloadEntryIdentifier":"3D7A0088-5ADF-4DF5-AD8C-
79ACCBDF6E38";"DownloadEntryRemoveWhenDoneKey":"false";"DownloadEntryPath":"/Users/tzworks/Downlo
ads/python-3.9.0-macosx10.9.pkg";"DownloadEntryShouldUseRequestURLAsOriginURLIfNecessaryKey":"false"}
```

```
extra fields
{"DownloadEntryDateAddedKey":"11/05/2020 18:31:48";"DownloadEntryDateFinishedKey":"11/05/2020
18:31:52";"DownloadEntryProgressBytesSoFar":"0x1db0a9c";"DownloadEntryProgressTotalToLoad":"0x1db0a9c";"DownloadEntryBookmarkBlob
":"62-6f-6f-6b-10-03-00-00-00-00-04-10-30-00-00-00-6b-09-77-23-4a-c4-fe-dc-a6-bb-58-bb-31-06-72-37-36-34-70-99-f2-86-66-2e-9b-8f-b6-06-4b-6c-
7c-9f-0c-02-00-00-04-00-00-00-03-03-00-00-00-08-00-28-05-00-00-00-01-01-00-00-55-73-65-72-73-00-00-00-07-00-00-00-01-01-00-00-74-7a-77-6f-72-
6b-73-00-09-00-00-00-01-01-00-00-44-6f-77-6e-6c-6f-61-64-73-00-00-00-24-00-00-00-01-01-00-00-70-79-74-68-6f-6e-2d-33-2e-39-2e-30-2d-6d-61-
63-6f-73-78-31-30-2e-39-2e-70-6b-67-2e-64-6f-77-6e-6c-6f-61-64-1b-00-00-00-01-01-00-00-70-79-74-68-6f-6e-2d-33-2e-39-2e-30-2d-6d-61-63-6f-
73-78-31-30-2e-39-2e-70-6b-67-00-14-00-00-00-01-06-00-00-10-00-00-00-20-00-00-00-30-00-00-00-44-00-00-00-70-00-00-00-08-00-00-00-04-03-00-
00-92-53-00-00-00-00-00-00-08-00-00-00-04-03-00-00-69-80-05-00-00-00-00-00-08-00-00-00-04-03-00-00-9c-80-05-00-00-00-00-00-08-00-00-00-04-
03-00-00-94-a0-48-00-00-00-00-00-08-00-00-00-04-03-00-00-97-a0-48-00-00-00-00-00-14-00-00-00-01-06-00-00-b0-00-00-00-c0-00-00-00-d0-00-00-
00-e0-00-00-00-f0-00-00-00-08-00-00-00-00-04-00-00-41-c2-aa-3e-4b-c3-a5-f4-18-00-00-00-01-02-00-00-01-00-00-00-00-00-00-00-0f-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-08-00-00-00-04-03-00-00-03-00-00-00-00-00-00-00-04-00-00-00-03-03-00-00-f5-01-00-00-08-00-00-00-01-09-00-00-
66-69-6c-65-3a-2f-2f-2f-0c-00-00-00-01-01-00-00-4d-61-63-69-6e-74-6f-73-68-20-48-44-08-00-00-00-04-03-00-00-00-70-c4-d8-d1-01-00-00-08-00-00-
00-00-04-00-00-41-c1-b0-a5-b4-33-31-40-24-00-00-00-01-01-00-00-46-33-41-30-41-32-41-36-2d-38-38-45-33-2d-34-32-37-30-2d-42-44-43-45-2d-43-
37-42-37-33-33-30-37-31-39-38-39-18-00-00-00-01-02-00-00-81-00-00-00-01-00-00-00-ef-13-00-00-01-00-00-00-00-00-00-00-00-00-00-00-01-00-00-
00-01-01-00-00-2f-00-00-00-00-00-00-00-01-05-00-00-cc-00-00-00-fe-ff-ff-ff-01-00-00-00-00-00-00-00-10-00-00-00-04-10-00-00-94-00-00-00-00-00-
00-00-05-10-00-00-00-01-00-00-00-00-00-00-10-10-00-00-2c-01-00-00-00-00-00-00-40-10-00-00-1c-01-00-00-00-00-00-00-02-20-00-00-f8-01-00-00-00-
00-00-00-05-20-00-00-68-01-00-00-00-00-00-00-10-20-00-00-78-01-00-00-00-00-00-00-11-20-00-00-ac-01-00-00-00-00-00-00-12-20-00-00-8c-01-00-
00-00-00-00-00-13-20-00-00-9c-01-00-00-00-00-00-00-20-20-00-00-d8-01-00-00-00-00-00-00-30-20-00-00-04-02-00-00-00-00-00-00-01-c0-00-00-4c-
01-00-00-00-00-00-00-11-c0-00-00-20-00-00-00-00-00-00-00-12-c0-00-00-5c-01-00-00-00-00-00-00-10-d0-00-00-04-00-00-00-00-00-00-
00";"DownloadEntryIdentifier":"3D7A0088-5ADF-4DF5-AD8C-
79ACCBDF6E38";"DownloadEntryRemoveWhenDoneKey":"false";"DownloadEntryPath":"/Users/tzworks/Downloads/python-3.9.0-
macosx10.9.pkg";"DownloadEntryShouldUseRequestURLAsOriginURLIfNecessaryKey":"false"}
```

4.3 *TopSites.plist*

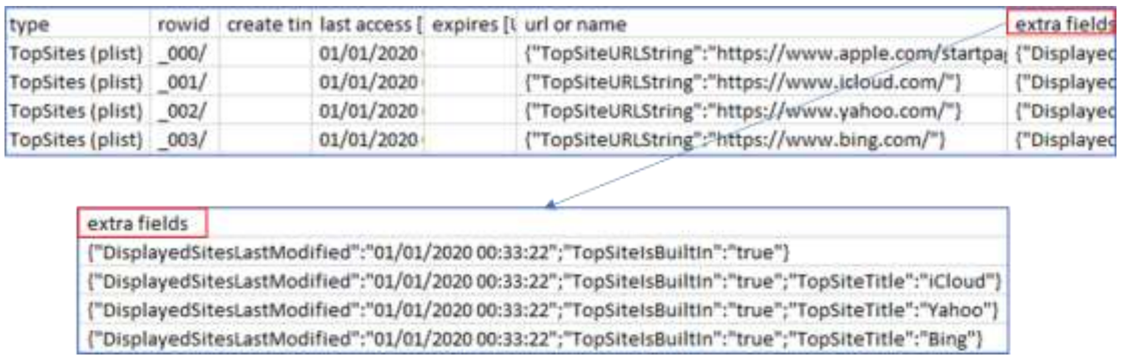
This *plist* is used in both the older and new *Safari* browsers and contains the *most visited* websites. This database stores the *URL* of the website visited, the websites title, and the last time it was updated.

The *TopSites.plist* file is located in the *<user's subdir>/Library/Safari/* folder. Below is the structure of this *plist* file.



Key	Type	Value
Root	Dictionary	(4 items)
TopSites	Array	(12 items)
Item 0	Dictionary	(2 items)
Item 1	Dictionary	(3 items)
TopSiteIsBuiltIn	Boolean	1
TopSiteURLString	String	https://www.icloud.com
TopSiteTitle	String	iCloud
Item 2	Dictionary	(3 items)
Item 3	Dictionary	(3 items)
DisplayedSitesLastModified	Date	2020-01-01T00:33:22
DemoSites	Array	(0 items)
BannedURLStrings	Array	(0 items)

Below is a sample output from the fields in the *TopSites.plist* database. Aside from the single timestamp that references the *last modification*, there contains no timestamp for the individual entries.



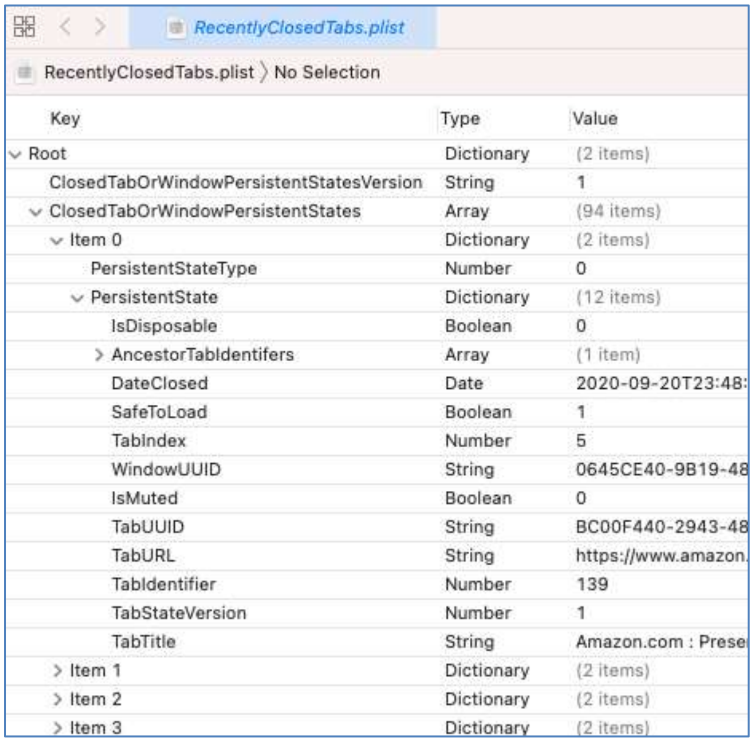
type	rowid	create tin	last access [	expires [l	url or name	extra fields
TopSites (plist)	_000/		01/01/2020		{"TopSiteURLString":"https://www.apple.com/startpa	{"Displayed
TopSites (plist)	_001/		01/01/2020		{"TopSiteURLString":"https://www.icloud.com/"}	{"Displayed
TopSites (plist)	_002/		01/01/2020		{"TopSiteURLString":"https://www.yahoo.com/"}	{"Displayed
TopSites (plist)	_003/		01/01/2020		{"TopSiteURLString":"https://www.bing.com/"}	{"Displayed

extra fields
{"DisplayedSitesLastModified":"01/01/2020 00:33:22";"TopSiteIsBuiltIn":"true"}
{"DisplayedSitesLastModified":"01/01/2020 00:33:22";"TopSiteIsBuiltIn":"true";"TopSiteTitle":"iCloud"}
{"DisplayedSitesLastModified":"01/01/2020 00:33:22";"TopSiteIsBuiltIn":"true";"TopSiteTitle":"Yahoo"}
{"DisplayedSitesLastModified":"01/01/2020 00:33:22";"TopSiteIsBuiltIn":"true";"TopSiteTitle":"Bing"}

4.4 *RecentlyClosedTabs.plist*

This *plist* is used in both the older and new *Safari* browsers and contains the browser tabs that were recently closed. This database stores the URL of the website visited, the tab title, and the time the tab was closed.

The *RecentlyClosedTabs.plist* file is located in the *<user's subdir>/Library/Safari/* folder. Below is the structure of this *plist* file.

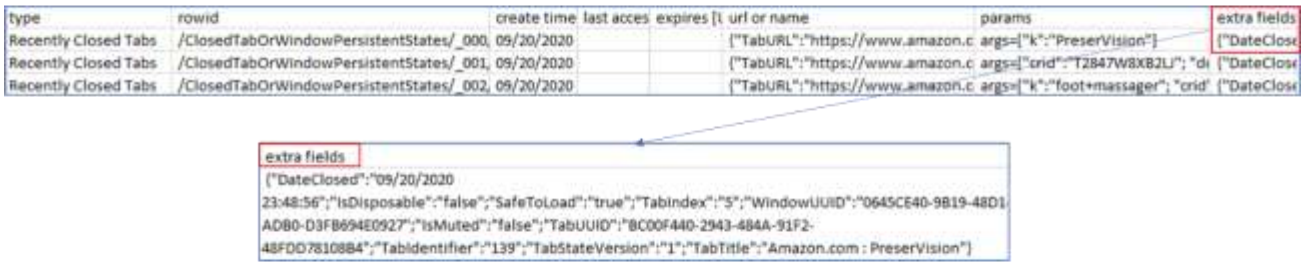


RecentlyClosedTabs.plist

RecentlyClosedTabs.plist 〉 No Selection

Key	Type	Value
Root	Dictionary	(2 items)
ClosedTabOrWindowPersistentStatesVersion	String	1
ClosedTabOrWindowPersistentStates	Array	(94 items)
Item 0	Dictionary	(2 items)
PersistentStateType	Number	0
PersistentState	Dictionary	(12 items)
IsDisposable	Boolean	0
AncestorTabIdentifers	Array	(1 item)
DateClosed	Date	2020-09-20T23:48:
SafeToLoad	Boolean	1
TabIndex	Number	5
WindowUUID	String	0645CE40-9B19-48
IsMuted	Boolean	0
TabUUID	String	BC00F440-2943-48
TabURL	String	https://www.amazon.
TabIdentifier	Number	139
TabStateVersion	Number	1
TabTitle	String	Amazon.com : Prese
Item 1	Dictionary	(2 items)
Item 2	Dictionary	(2 items)
Item 3	Dictionary	(2 items)

Below is a sample output from the fields in the *RecentlyClosedTabs.plist* database. The '*extra fields*' column is expanded for the first entry as well.



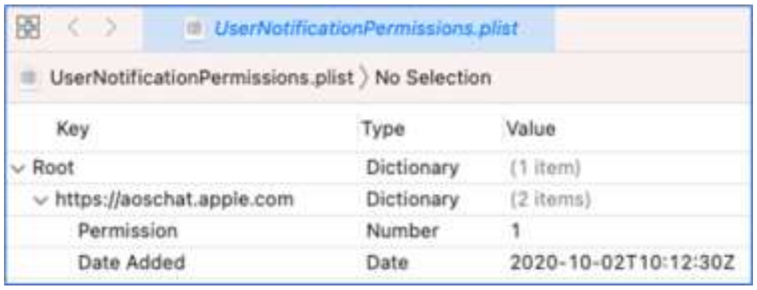
type	rowid	create time	last acces	expires [t	url or name	params	extra fields
Recently Closed Tabs	/ClosedTabOrWindowPersistentStates/_000,	09/20/2020			{"TabURL":"https://www.amazon.c	args=["k":"PreserVision"]	{"DateClose
Recently Closed Tabs	/ClosedTabOrWindowPersistentStates/_001,	09/20/2020			{"TabURL":"https://www.amazon.c	args=["crid":"T2847W8XB2LI"; "d	{"DateClose
Recently Closed Tabs	/ClosedTabOrWindowPersistentStates/_002,	09/20/2020			{"TabURL":"https://www.amazon.c	args=["k":"foot+massager"; "crid'	{"DateClose

```
extra fields
{"DateClosed":"09/20/2020
23:48:56";"IsDisposable":"false";"SafeToLoad":"true";"TabIndex":"5";"WindowUUID":"0645CE40-9B19-48D1-
ADB0-D3FB694E0927";"IsMuted":"false";"TabUUID":"BC00F440-2943-484A-91F2-
48FDD78108B4";"TabIdentifier":"139";"TabStateVersion":"1";"TabTitle":"Amazon.com : PreserVision"}
```

4.5 *UserNotificationPermissions.plist*

This *plist* is used in both the older and new *Safari* browsers and contains the specific permissions associated with a URL/domain. This database stores the *URL*, the permission in the form of a number, and the time the permission was added.

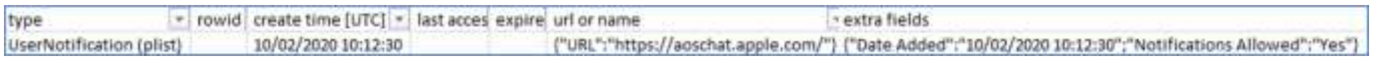
The *UserNotificationPermissions.plist* file is located in the *<user's subdir>/Library/Safari/* folder. Below is the structure of this *plist* file.



UserNotificationPermissions.plist 〉 No Selection

Key	Type	Value
Root	Dictionary	(1 item)
https://aoschat.apple.com	Dictionary	(2 items)
Permission	Number	1
Date Added	Date	2020-10-02T10:12:30Z

Below is a sample output from the fields in the *UserNotificationPermissions.plist* database.

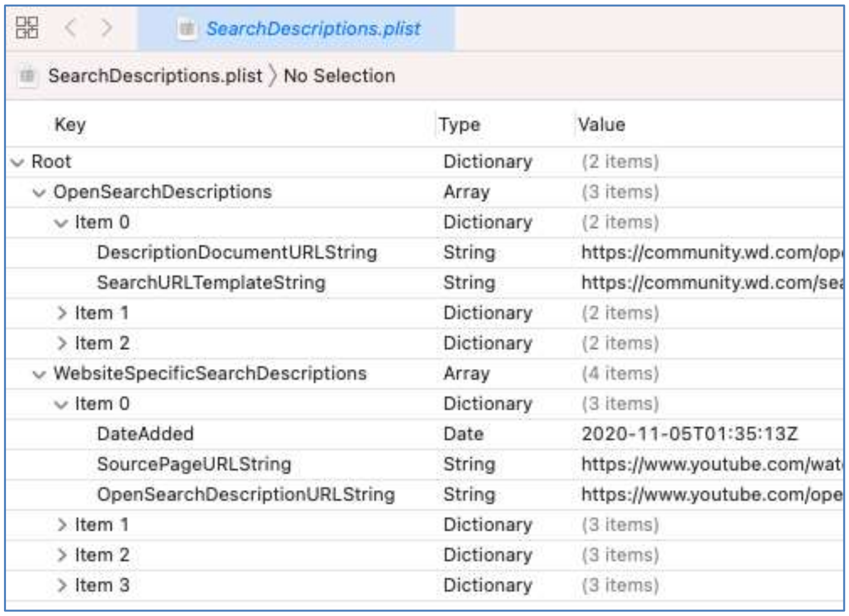


type	rowid	create time [UTC]	last acces	expire	url or name	extra fields
UserNotification (plist)		10/02/2020 10:12:30			{"URL":"https://aoschat.apple.com/"}	{"Date Added":"10/02/2020 10:12:30";"Notifications Allowed":"Yes"}

4.6 *SearchDescriptions.plist*

This *plist* is used in both the older and new *Safari* browsers and contains the search description used. This database stores the URL, and in *WebsiteSpecificSearchDescriptions* listing, the timestamp that it was added.

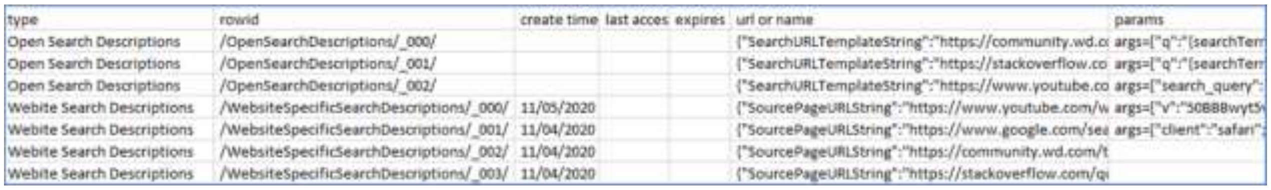
The *SearchDescriptions.plist* file is located in the *<user's subdir>/Library/Safari/* folder. Below is the structure of this *plist* file.



SearchDescriptions.plist 〉 No Selection

Key	Type	Value
Root	Dictionary	(2 items)
OpenSearchDescriptions	Array	(3 items)
Item 0	Dictionary	(2 items)
DescriptionDocumentURLString	String	https://community.wd.com/op
SearchURLTemplateString	String	https://community.wd.com/sea
Item 1	Dictionary	(2 items)
Item 2	Dictionary	(2 items)
WebsiteSpecificSearchDescriptions	Array	(4 items)
Item 0	Dictionary	(3 items)
DateAdded	Date	2020-11-05T01:35:13Z
SourcePageURLString	String	https://www.youtube.com/wat
OpenSearchDescriptionURLString	String	https://www.youtube.com/ope
Item 1	Dictionary	(3 items)
Item 2	Dictionary	(3 items)
Item 3	Dictionary	(3 items)

Below is a sample output from the fields in the *SearchDescriptions.plist* database. Notice that this artifact contains 2 categories: (a) *OpenSearchDescriptions* and the related (b) *WebsiteSpecificSearchDescriptions*. For the purposes of the output, each category is broken out separately.

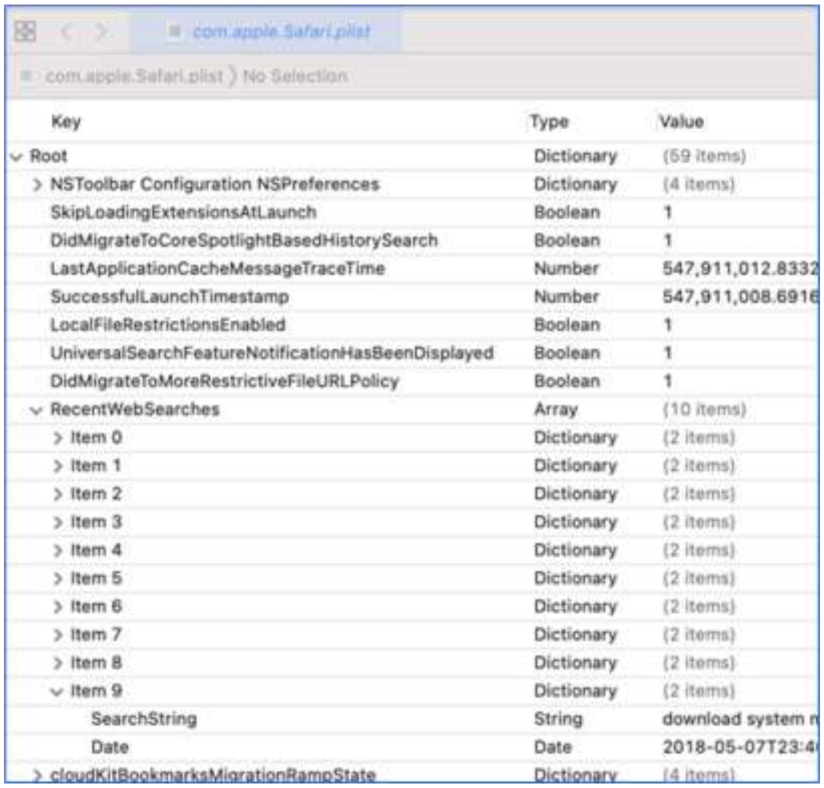


type	rowid	create time	last acces	expires	url or name	params
Open Search Descriptions	/OpenSearchDescriptions/_000/				{"SearchURLTemplateString":"https://community.wd.c	args=["q":"{searchTem
Open Search Descriptions	/OpenSearchDescriptions/_001/				{"SearchURLTemplateString":"https://stackoverflow.co	args=["q":"{searchTem
Open Search Descriptions	/OpenSearchDescriptions/_002/				{"SearchURLTemplateString":"https://www.youtube.co	args=["search_query":
Webite Search Descriptions	/WebsiteSpecificSearchDescriptions/_000/	11/05/2020			{"SourcePageURLString":"https://www.youtube.com/w	args=["v":"50BBBwyt5
Webite Search Descriptions	/WebsiteSpecificSearchDescriptions/_001/	11/04/2020			{"SourcePageURLString":"https://www.google.com/sea	args=["client":"safari":
Webite Search Descriptions	/WebsiteSpecificSearchDescriptions/_002/	11/04/2020			{"SourcePageURLString":"https://community.wd.com/t	
Webite Search Descriptions	/WebsiteSpecificSearchDescriptions/_003/	11/04/2020			{"SourcePageURLString":"https://stackoverflow.com/q	

4.7 *com.apple.Safari.plist*

This *plist* is used for *Safari* preferences, and thus contains a list of settings as well as some timestamps for certain items. The data of key/values is displayed in the '*extra fields*' in the output since it was not conducive to fit into any of the other fields.

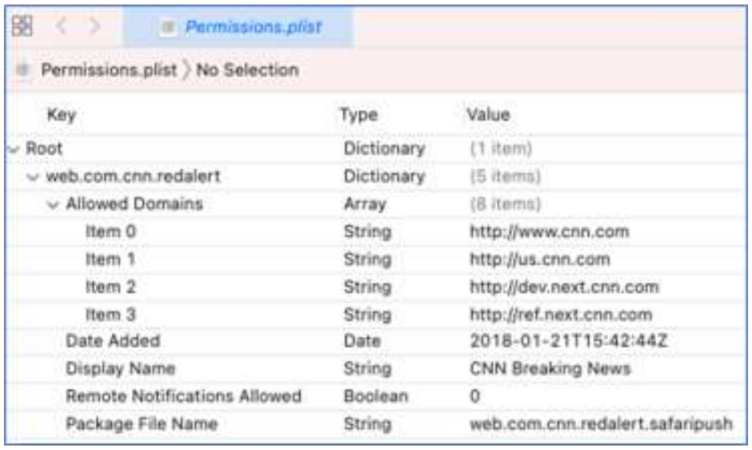
The *com.apple.Safari.plist* file is either located in the *<user's subdir>/Library/Preferences/* folder. Below is the structure of this *plist* file. It contains many *key/value* pairs mostly unrelated to one another. There is however, a section on *RecentWebSearches* that include the search string and the timestamp.



Key	Type	Value
Root	Dictionary	(59 items)
NSToolbar Configuration NSPreferences	Dictionary	(4 items)
SkipLoadingExtensionsAtLaunch	Boolean	1
DidMigrateToCoreSpotlightBasedHistorySearch	Boolean	1
LastApplicationCacheMessageTraceTime	Number	547,911,012.8332
SuccessfulLaunchTimestamp	Number	547,911,008.6916
LocalFileRestrictionsEnabled	Boolean	1
UniversalSearchFeatureNotificationHasBeenDisplayed	Boolean	1
DidMigrateToMoreRestrictiveFileURLPolicy	Boolean	1
RecentWebSearches	Array	(10 items)
Item 0	Dictionary	(2 items)
Item 1	Dictionary	(2 items)
Item 2	Dictionary	(2 items)
Item 3	Dictionary	(2 items)
Item 4	Dictionary	(2 items)
Item 5	Dictionary	(2 items)
Item 6	Dictionary	(2 items)
Item 7	Dictionary	(2 items)
Item 8	Dictionary	(2 items)
Item 9	Dictionary	(2 items)
SearchString	String	download system n
Date	Date	2018-05-07T23:4
cloudKitBookmarksMigrationRampState	Dictionary	(4 items)

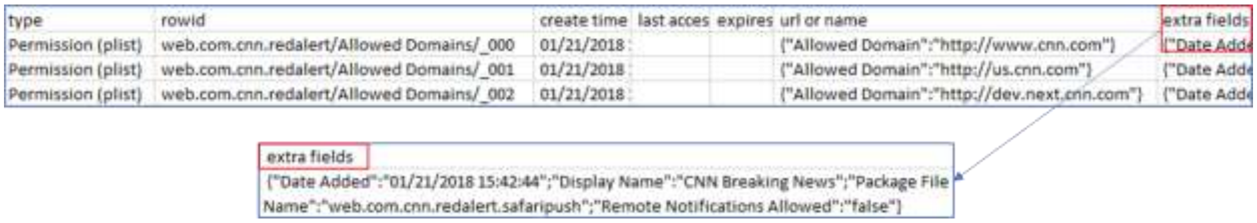
4.8 *Permissions.plist*

The *Permissions.plist* file is located in the *<user's subdir>/Library/Safari/RemoteNotifications/* folder. Below is the structure of this *plist* file.



Key	Type	Value
Root	Dictionary	(1 item)
web.com.cnn.redalert	Dictionary	(5 items)
Allowed Domains	Array	(8 items)
Item 0	String	http://www.cnn.com
Item 1	String	http://us.cnn.com
Item 2	String	http://dev.next.cnn.com
Item 3	String	http://ref.next.cnn.com
Date Added	Date	2018-01-21T15:42:44Z
Display Name	String	CNN Breaking News
Remote Notifications Allowed	Boolean	0
Package File Name	String	web.com.cnn.redalert.safaripush

Below is a sample output from the fields in the *Permissions.plist* database. There is one timestamp (*Date Added*) per dictionary item, which contains an array of domain names. Each line in the output is one domain name with the associated '*common*' metadata that goes with the dictionary item (which in this example is *web.com.cnn.redalert*).

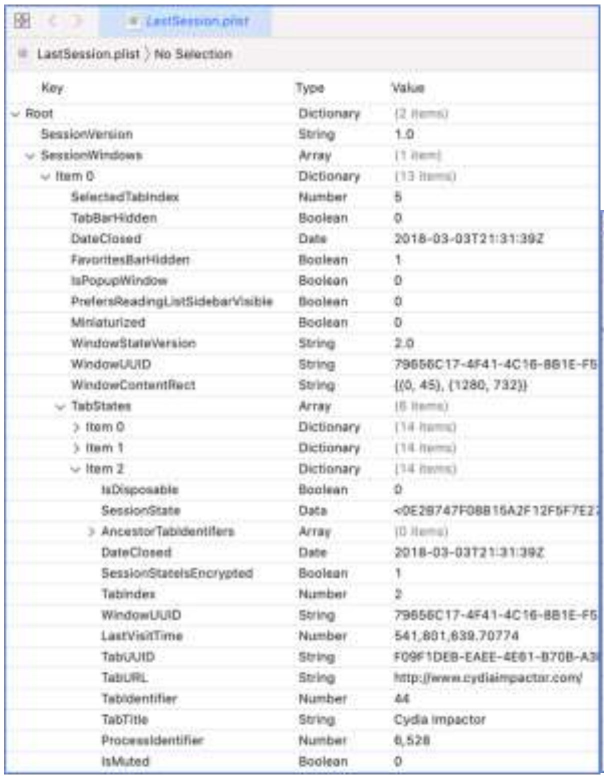


type	rowid	create time	last acces	expires	url or name	extra fields
Permission (plist)	web.com.cnn.redalert/Allowed Domains/_000	01/21/2018			{"Allowed Domain":"http://www.cnn.com"}	{"Date Adde
Permission (plist)	web.com.cnn.redalert/Allowed Domains/_001	01/21/2018			{"Allowed Domain":"http://us.cnn.com"}	{"Date Adde
Permission (plist)	web.com.cnn.redalert/Allowed Domains/_002	01/21/2018			{"Allowed Domain":"http://dev.next.cnn.com"}	{"Date Adde

extra fields
{"Date Added":"01/21/2018 15:42:44";"Display Name":"CNN Breaking News";"Package File Name":"web.com.cnn.redalert.safaripush";"Remote Notifications Allowed":"false"}

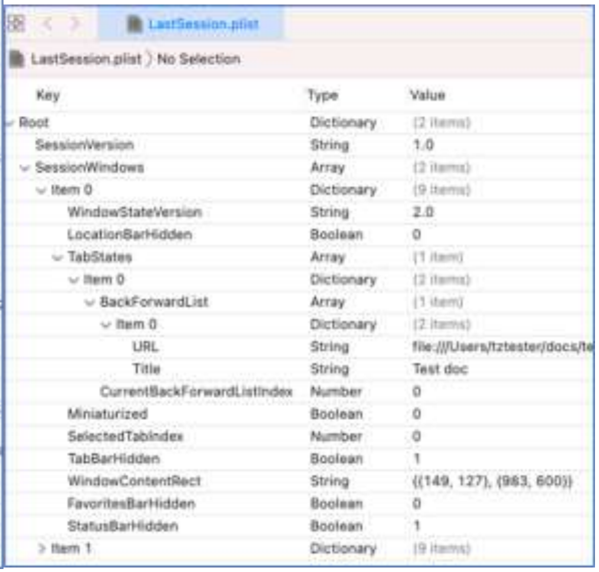
4.9 *LastSession.plist*

The *LastSession.plist* file is located in the *<user's subdir>/Library/Safari/* folder. The structure of the data is shown below. While some of the URLs are enumerated directly in each '*TabStates*' item index, there are cases where the URLs are nested in the '*BackForwardList'* element of the item index. Both are shown below.



LastSession.plist 〉 No Selection

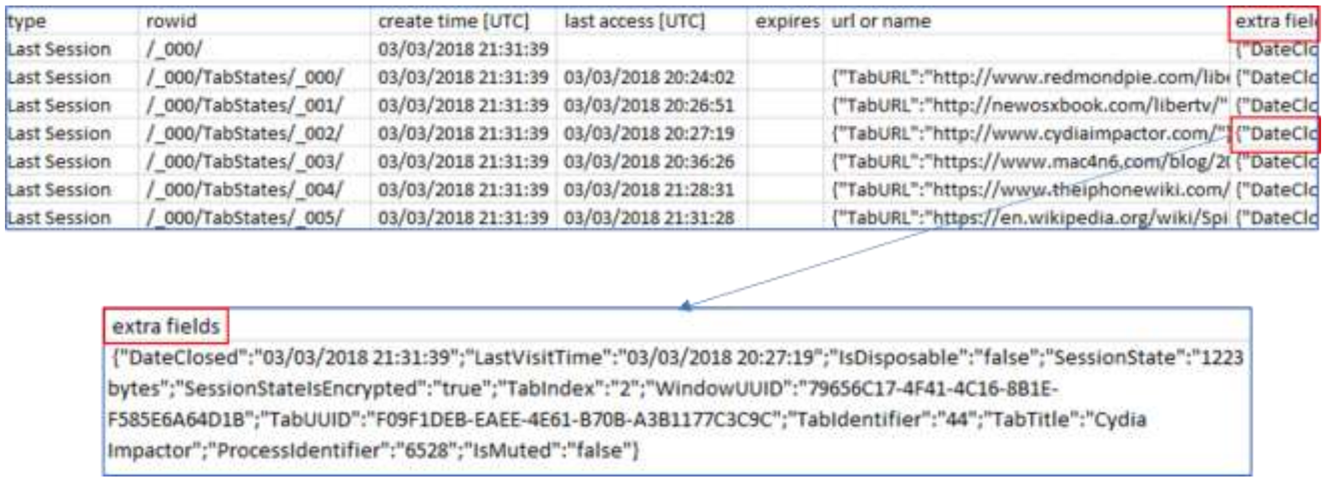
Key	Type	Value
Root	Dictionary	(2 items)
SessionVersion	String	1.0
SessionWindows	Array	(1 item)
Item 0	Dictionary	(13 items)
SelectedTabIndex	Number	5
TabBarHidden	Boolean	0
DateClosed	Date	2018-03-03T21:31:39Z
FavoritesBarHidden	Boolean	1
IsPopupWindow	Boolean	0
PrefersReadingListSidebarVisible	Boolean	0
Miniaturized	Boolean	0
WindowStateVersion	String	2.0
WindowUUID	String	79656C17-4F41-4C16-8B1E-F5
WindowContentRect	String	{{0, 45}, {1280, 732}}
TabStates	Array	(6 items)
Item 0	Dictionary	(14 items)
Item 1	Dictionary	(14 items)
Item 2	Dictionary	(14 items)
IsDisposable	Boolean	0
SessionState	Data	<0E28747F08B16A2F12F5F7E2
AncestorTabIdentifiers	Array	(0 items)
DateClosed	Date	2018-03-03T21:31:39Z
SessionStateIsEncrypted	Boolean	1
TabIndex	Number	2
WindowUUID	String	79656C17-4F41-4C16-8B1E-F5
LastVisitTime	Number	541,801,839.70774
TabUUID	String	F09F1DEB-EAEE-4E61-B70B-A3
TabURL	String	http://www.cydiaimpactor.com/
TabIdentifier	Number	44
TabTitle	String	Cydia Impactor
ProcessIdentifier	Number	6,528
IsMuted	Boolean	0



LastSession.plist 〉 No Selection

Key	Type	Value
Root	Dictionary	(2 items)
SessionVersion	String	1.0
SessionWindows	Array	(2 items)
Item 0	Dictionary	(9 items)
WindowStateVersion	String	2.0
LocationBarHidden	Boolean	0
TabStates	Array	(1 item)
Item 0	Dictionary	(2 items)
BackForwardList	Array	(1 item)
Item 0	Dictionary	(2 items)
URL	String	file:///Users/tztester/docs/te
Title	String	Test doc
CurrentBackForwardListIndex	Number	0
Miniaturized	Boolean	0
SelectedTabIndex	Number	0
TabBarHidden	Boolean	1
WindowContentRect	String	{{149, 127}, {983, 600}}
FavoritesBarHidden	Boolean	0
StatusBarHidden	Boolean	1
Item 1	Dictionary	(9 items)

Below is a sample output from the fields in the *LastSession.plist* database. The '*create time*' column shows the '*DateClosed*' timestamp. The details of the other metadata aside from timestamps and URL, are shown in the '*extra fields*' column.



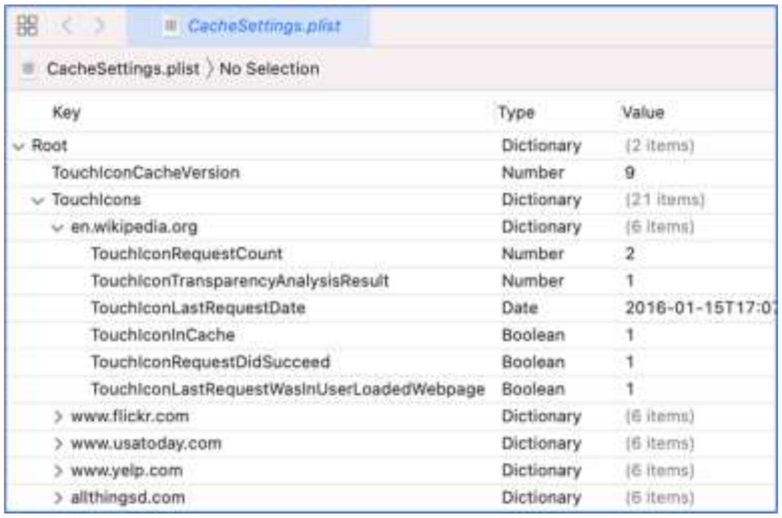
type	rowid	create time [UTC]	last access [UTC]	expires	url or name	extra fiel
Last Session	/_000/	03/03/2018 21:31:39				{"DateClo
Last Session	/_000/TabStates/_000/	03/03/2018 21:31:39	03/03/2018 20:24:02		{"TabURL":"http://www.redmondpie.com/libe	{"DateClo
Last Session	/_000/TabStates/_001/	03/03/2018 21:31:39	03/03/2018 20:26:51		{"TabURL":"http://newosxbook.com/libertv/"	{"DateClo
Last Session	/_000/TabStates/_002/	03/03/2018 21:31:39	03/03/2018 20:27:19		{"TabURL":"http://www.cydiaimpactor.com/"	{"DateClo
Last Session	/_000/TabStates/_003/	03/03/2018 21:31:39	03/03/2018 20:36:26		{"TabURL":"https://www.mac4n6.com/blog/2(	{"DateClo
Last Session	/_000/TabStates/_004/	03/03/2018 21:31:39	03/03/2018 21:28:31		{"TabURL":"https://www.theiphonewiki.com/	{"DateClo
Last Session	/_000/TabStates/_005/	03/03/2018 21:31:39	03/03/2018 21:31:28		{"TabURL":"https://en.wikipedia.org/wiki/Spi	{"DateClo

extra fields

{"DateClosed":"03/03/2018 21:31:39";"LastVisitTime":"03/03/2018 20:27:19";"IsDisposable":"false";"SessionState":"1223 bytes";"SessionStateIsEncrypted":"true";"TabIndex":"2";"WindowUUID":"79656C17-4F41-4C16-8B1E-F585E6A64D1B";"TabUUID":"F09F1DEB-EAEE-4E61-B70B-A3B1177C3C9C";"TabIdentifier":"44";"TabTitle":"Cydia Impactor";"ProcessIdentifier":"6528";"IsMuted":"false"}

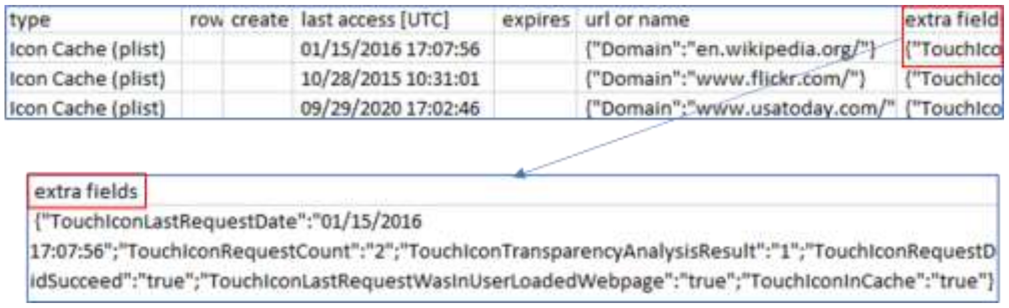
4.10 *CacheSettings.plist*

The *CacheSettings.plist* file is located in the *<user's subdir>/Library/Safari/Touch Icons/* folder. It is found in the older versions of *Safari*; the newer artifact is the *TouchIconCacheSettings.db* which was discussed previously in the *SQLite* artifact section. The structure of this *plist* type is shown below.



Key	Type	Value
Root	Dictionary	(2 items)
TouchIconCacheVersion	Number	9
TouchIcons	Dictionary	(21 items)
en.wikipedia.org	Dictionary	(6 items)
TouchIconRequestCount	Number	2
TouchIconTransparencyAnalysisResult	Number	1
TouchIconLastRequestDate	Date	2016-01-15T17:0
TouchIconInCache	Boolean	1
TouchIconRequestDidSucceed	Boolean	1
TouchIconLastRequestWasInUserLoadedWebpage	Boolean	1
www.flickr.com	Dictionary	(6 items)
www.usatoday.com	Dictionary	(6 items)
www.yelp.com	Dictionary	(6 items)
allthingsd.com	Dictionary	(6 items)

Below is a sample output from the fields in the *CacheSettings.plist* database. The '*last access time*' column shows the '*TouchIconLastRequestDate*' timestamp. The other metadata aside from timestamp and domain, are shown in the '*extra fields*' column.



type	row	create	last access [UTC]	expires	url or name	extra field
Icon Cache (plist)			01/15/2016 17:07:56		{"Domain":"en.wikipedia.org/"}	{"TouchIco
Icon Cache (plist)			10/28/2015 10:31:01		{"Domain":"www.flickr.com/"}	{"TouchIco
Icon Cache (plist)			09/29/2020 17:02:46		{"Domain":"www.usatoday.com/"	{"TouchIco

extra fields
{"TouchIconLastRequestDate":"01/15/2016 17:07:56";"TouchIconRequestCount":"2";"TouchIconTransparencyAnalysisResult":"1";"TouchIconRequestDidSucceed":"true";"TouchIconLastRequestWasInUserLoadedWebpage":"true";"TouchIconInCache":"true"}

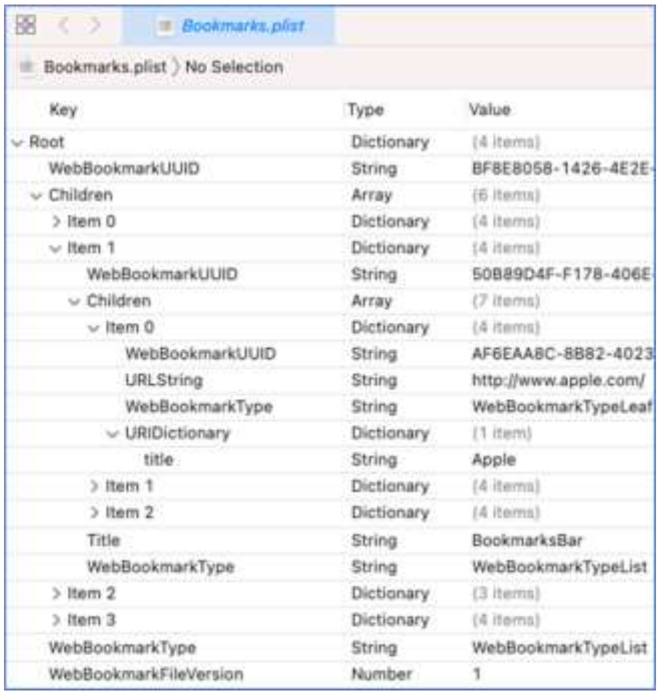
4.11 Miscellaneous plist files

These are a category of *plist* files that are parsed by ***sap*** but the fields in these *plist* files do not contain any explicit timestamps. Therefore, when the tool parses these specific files, the output would not show up in timestamp dependent output such as *Log2Timeline* using the option ***-csvl2t***. For these files, therefore, one can only use the default output or the ***-csv*** option.

4.11.1 *Bookmarks.plist*

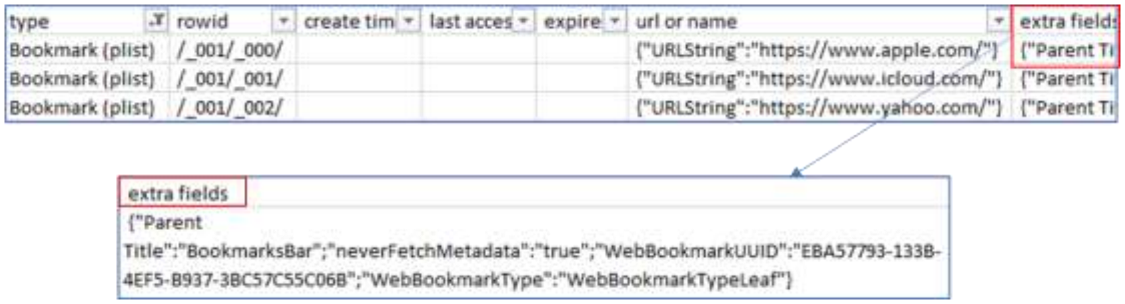
This *plist* is used in both the older and new *Safari* browsers and stores the configuration of the bookmarks for the browser. It has the *URL* of the website, the *parent bookmark title*, the *item's title* and various other *universally unique identifiers* (UUIDs). There are no timestamp fields identifying when the bookmark was added.

The *Bookmarks.plist* file is located in the *<user's subdir>/Library/Safari/* folder. Below is the structure of this *plist* file.



Key	Type	Value
Root	Dictionary	(4 items)
WebBookmarkUUID	String	BF8E8058-1426-4E2E-
Children	Array	(6 items)
Item 0	Dictionary	(4 items)
Item 1	Dictionary	(4 items)
WebBookmarkUUID	String	50B89D4F-F178-406E-
Children	Array	(7 items)
Item 0	Dictionary	(4 items)
WebBookmarkUUID	String	AF6EAA8C-8B82-4023
URLString	String	http://www.apple.com/
WebBookmarkType	String	WebBookmarkTypeLeaf
URIDictionary	Dictionary	(1 item)
title	String	Apple
Item 1	Dictionary	(4 items)
Item 2	Dictionary	(4 items)
Title	String	BookmarksBar
WebBookmarkType	String	WebBookmarkTypeList
Item 2	Dictionary	(3 items)
Item 3	Dictionary	(4 items)
WebBookmarkType	String	WebBookmarkTypeList
WebBookmarkFileVersion	Number	1

Below is a sample output from the fields in the *Bookmarks.plist* database.



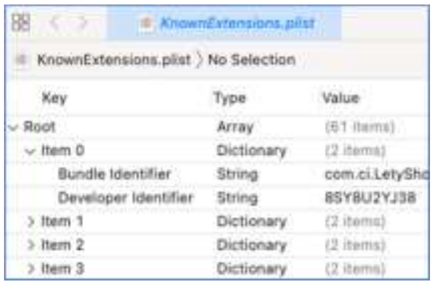
type	rowid	create tim	last acces	expire	url or name	extra field
Bookmark (plist)	/_001/_000/				{"URLString":"https://www.apple.com/"}	{"Parent Ti
Bookmark (plist)	/_001/_001/				{"URLString":"https://www.icloud.com/"}	{"Parent Ti
Bookmark (plist)	/_001/_002/				{"URLString":"https://www.yahoo.com/"}	{"Parent Ti

extra fields
{"Parent Title":"BookmarksBar";"neverFetchMetadata":"true";"WebBookmarkUUID":"EBA57793-133B-4EF5-B937-3BC57C55C06B";"WebBookmarkType":"WebBookmarkTypeLeaf"}

4.11.2 *KnownExtensions.plist*

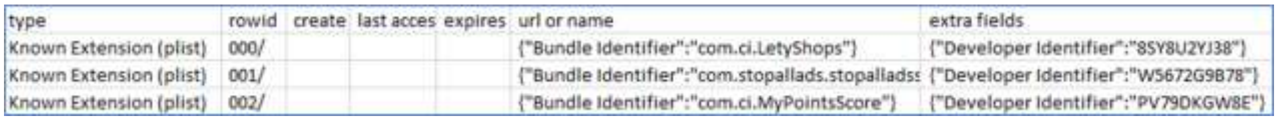
This *plist* is used in just the older *Safari* browsers and it is used to identify the application bundle and identifier of the developer.

The *KnownExtensions.plist* file is located in the *<user's subdir>/Library/Safari/* folder.   Below is the structure of this *plist* file.



Key	Type	Value
Root	Array	(61 items)
Item 0	Dictionary	(2 items)
Bundle Identifier	String	com.ci.LetySho
Developer Identifier	String	8SY8U2YJ38
Item 1	Dictionary	(2 items)
Item 2	Dictionary	(2 items)
Item 3	Dictionary	(2 items)

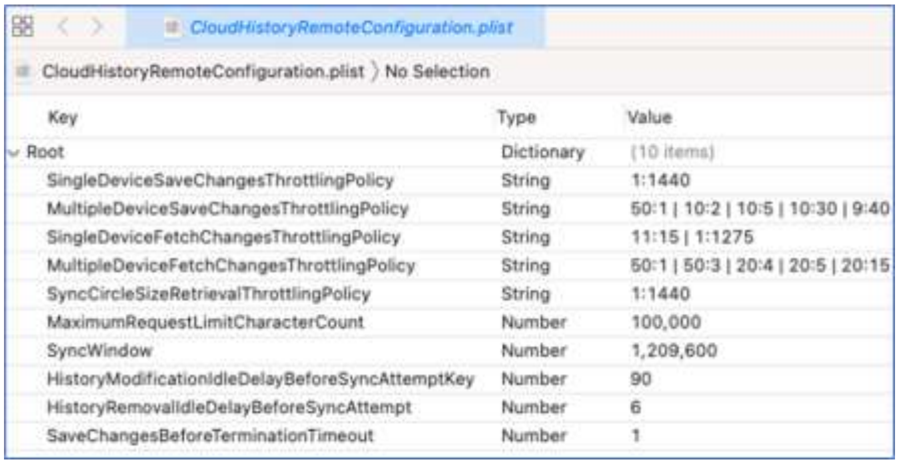
Below is a sample output from the fields in the *KnownExtensions.plist* database.



type	rowid	create	last acces	expires	url or name	extra fields
Known Extension (plist)	000/				{"Bundle Identifier":"com.ci.LetyShops"}	{"Developer Identifier":"8SY8U2YJ38"}
Known Extension (plist)	001/				{"Bundle Identifier":"com.stopallads.stopallads:	{"Developer Identifier":"W5672G9B78"}
Known Extension (plist)	002/				{"Bundle Identifier":"com.ci.MyPointsScore"}	{"Developer Identifier":"PV79DKGW8E"}

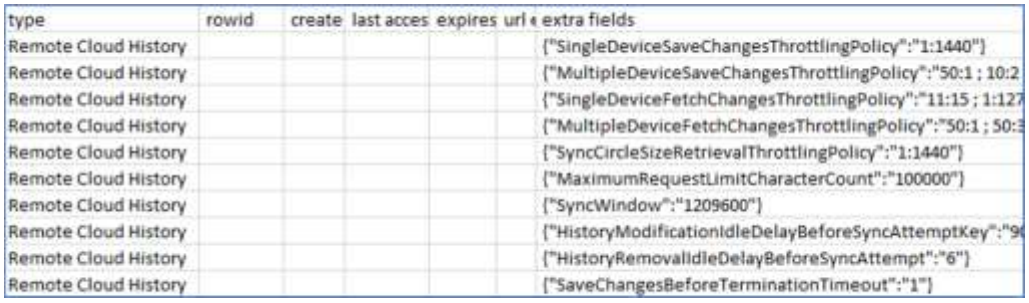
4.11.3 *CloudHistoryRemoteConfiguration.plist*

The *CloudHistoryRemoteConfiguration.plist* file is located in the *<user's subdir>/Library/Safari/* folder. Below is the structure of this *plist* file.


CloudHistoryRemoteConfiguration.plist 〉 No Selection

Key	Type	Value
Root	Dictionary	(10 items)
SingleDeviceSaveChangesThrottlingPolicy	String	1:1440
MultipleDeviceSaveChangesThrottlingPolicy	String	50:1 | 10:2 | 10:5 | 10:30 | 9:40
SingleDeviceFetchChangesThrottlingPolicy	String	11:15 | 1:1275
MultipleDeviceFetchChangesThrottlingPolicy	String	50:1 | 50:3 | 20:4 | 20:5 | 20:15
SyncCircleSizeRetrievalThrottlingPolicy	String	1:1440
MaximumRequestLimitCharacterCount	Number	100,000
SyncWindow	Number	1,209,600
HistoryModificationIdleDelayBeforeSyncAttemptKey	Number	90
HistoryRemovalIdleDelayBeforeSyncAttempt	Number	6
SaveChangesBeforeTerminationTimeout	Number	1

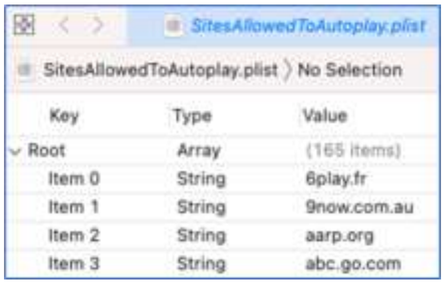
Below is a sample output from the fields in the *CloudHistoryRemoteConfiguration.plist* database.   There are no explicit timestamps or URLs in the record data. The data is strictly configuration data as its filename suggests. In this case, all the data *key/value* pairs are put into the '*extra fields*' column.



type	rowid	create	last acces	expires	url	extra fields
Remote Cloud History						{"SingleDeviceSaveChangesThrottlingPolicy":"1:1440"}
Remote Cloud History						{"MultipleDeviceSaveChangesThrottlingPolicy":"50:1 ; 10:2
Remote Cloud History						{"SingleDeviceFetchChangesThrottlingPolicy":"11:15 ; 1:127
Remote Cloud History						{"MultipleDeviceFetchChangesThrottlingPolicy":"50:1 ; 50:
Remote Cloud History						{"SyncCircleSizeRetrievalThrottlingPolicy":"1:1440"}
Remote Cloud History						{"MaximumRequestLimitCharacterCount":"100000"}
Remote Cloud History						{"SyncWindow":"1209600"}
Remote Cloud History						{"HistoryModificationIdleDelayBeforeSyncAttemptKey":"9
Remote Cloud History						{"HistoryRemovalIdleDelayBeforeSyncAttempt":"6"}
Remote Cloud History						{"SaveChangesBeforeTerminationTimeout":"1"}

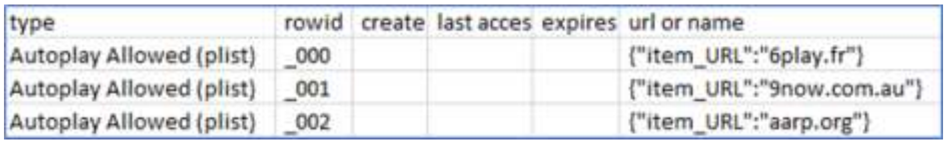
4.11.4 *SitesAllowedToAutoPlay.plist*

The *SitesAllowedToAutoPlay.plist* file is located in the *<user's subdir>/Library/Safari/* folder.   Below is the structure of this *plist* file. This *plist* only contains a listing of domains that are allowed to have auto-play enabled.


SitesAllowedToAutoplay.plist 〉 No Selection

Key	Type	Value
Root	Array	(165 items)
Item 0	String	6play.fr
Item 1	String	9now.com.au
Item 2	String	aarp.org
Item 3	String	abc.go.com

Below is a sample output from the fields in the *SitesAllowedToAutoPlay.plist* database.   Each domain is listed on a separate line in the output.



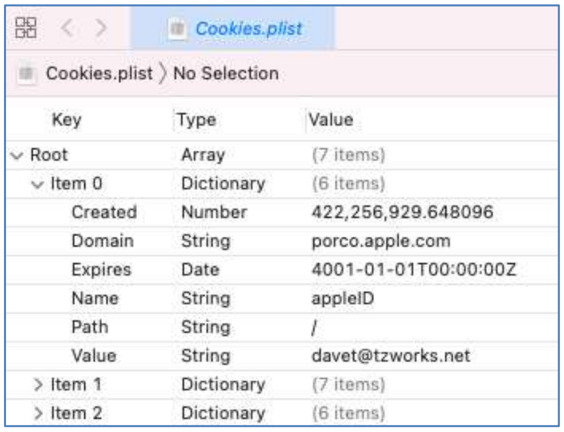
type	rowid	create	last acces	expires	url or name
Autoplay Allowed (plist)	_000				{"item_URL":"6play.fr"}
Autoplay Allowed (plist)	_001				{"item_URL":"9now.com.au"}
Autoplay Allowed (plist)	_002				{"item_URL":"aarp.org"}

5 Cookies

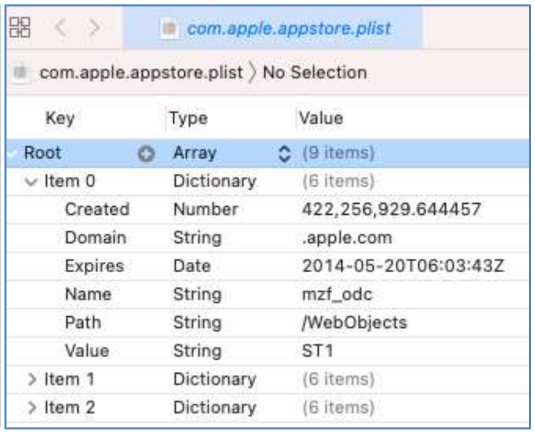
sap also parses *Safari* cookie files. Depending on whether the *Safari* browser artifacts are from the older or newer versions, these files can either be of the form: (a) *Cookies.plist* and *<reverse_domain_name>.plist*, or (b) *Cookies.binarycookies* and *<reverse_domain_name>.binarycookies* The first type are used in the older *Safari* browsers. The second type are found in the newer *Safari* versions. Both of these types of files are located in the *<user's subdir>/Library/Cookies* subfolder.

5.1 *Cookies.plist*

Below is the structure of the *plist* version of the cookie file. One can see the format for the *Cookies.plist* is the similar to the *<reverse-domain-name>.plist* files.

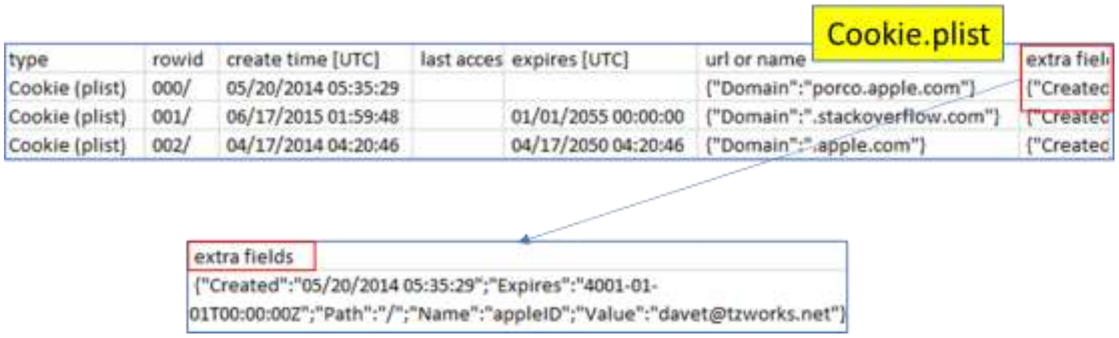


Key	Type	Value
Root	Array	(7 items)
Item 0	Dictionary	(6 items)
Created	Number	422,256,929.648096
Domain	String	porco.apple.com
Expires	Date	4001-01-01T00:00:00Z
Name	String	appleID
Path	String	/
Value	String	davet@tzworks.net
Item 1	Dictionary	(7 items)
Item 2	Dictionary	(6 items)



Key	Type	Value
Root	Array	(9 items)
Item 0	Dictionary	(6 items)
Created	Number	422,256,929.644457
Domain	String	.apple.com
Expires	Date	2014-05-20T06:03:43Z
Name	String	mzf_odc
Path	String	/WebObjects
Value	String	ST1
Item 1	Dictionary	(6 items)
Item 2	Dictionary	(6 items)

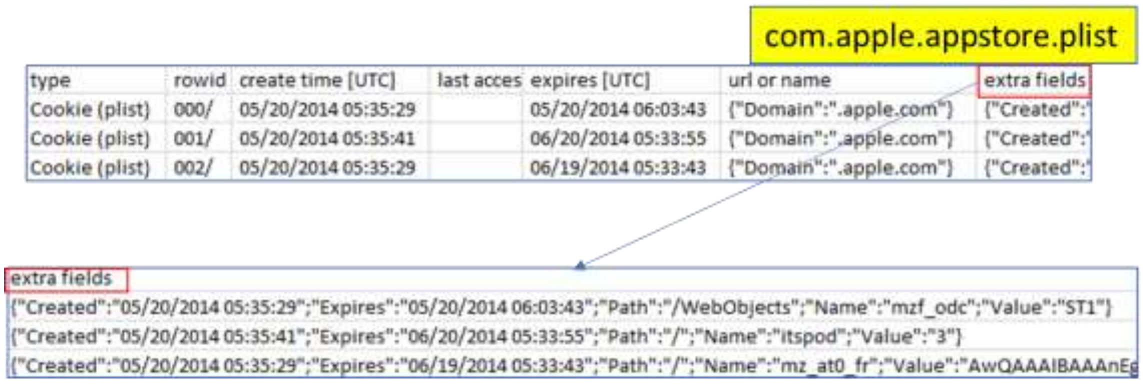
Below are the outputs for both *plist* file types. The data should be similar, since they both use the same structure.



Cookie.plist

type	rowid	create time [UTC]	last acces	expires [UTC]	url or name	extra fiel
Cookie (plist)	000/	05/20/2014 05:35:29			{"Domain":"porco.apple.com"}	{"Createc
Cookie (plist)	001/	06/17/2015 01:59:48		01/01/2055 00:00:00	{"Domain":".stackoverflow.com"}	{"Createc
Cookie (plist)	002/	04/17/2014 04:20:46		04/17/2050 04:20:46	{"Domain":".apple.com"}	{"Createc

extra fields
{"Created":"05/20/2014 05:35:29";"Expires":"4001-01-01T00:00:00Z";"Path":"/";"Name":"appleID";"Value":"davet@tzworks.net"}

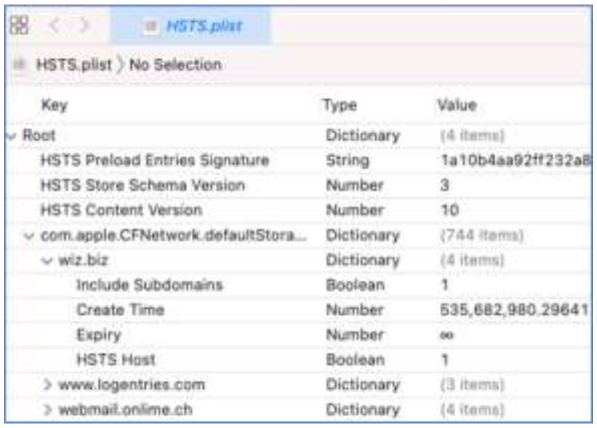

com.apple.appstore.plist

type	rowid	create time [UTC]	last acces	expires [UTC]	url or name	extra fields
Cookie (plist)	000/	05/20/2014 05:35:29		05/20/2014 06:03:43	{"Domain":".apple.com"}	{"Created":'
Cookie (plist)	001/	05/20/2014 05:35:41		06/20/2014 05:33:55	{"Domain":".apple.com"}	{"Created":'
Cookie (plist)	002/	05/20/2014 05:35:29		06/19/2014 05:33:43	{"Domain":".apple.com"}	{"Created":'

extra fields
{"Created":"05/20/2014 05:35:29";"Expires":"05/20/2014 06:03:43";"Path":"/WebObjects";"Name":"mzf_odc";"Value":"ST1"}
{"Created":"05/20/2014 05:35:41";"Expires":"06/20/2014 05:33:55";"Path":"/";"Name":"itspod";"Value":"3"}
{"Created":"05/20/2014 05:35:29";"Expires":"06/19/2014 05:33:43";"Path":"/";"Name":"mz_at0_fr";"Value":"AwQAAAIBAAAnEg

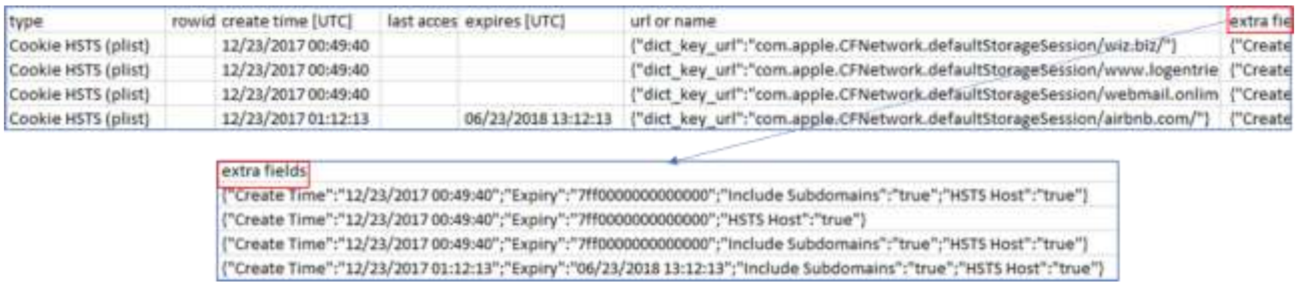
5.2 *HSTS.plist*

The *HSTS* in the filename is short for *HTTP Strict Transport Security Request* and is present on the new versions of *Safari*. This file is also located in the *<user's subdir>/Library/Cookies* folder. Below are the fields present in this *plist* file.



Key	Type	Value
Root	Dictionary	(4 items)
HSTS Preload Entries Signature	String	1a10b4aa92ff232a8
HSTS Store Schema Version	Number	3
HSTS Content Version	Number	10
com.apple.CFNetwork.defaultStora...	Dictionary	(744 items)
wiz.biz	Dictionary	(4 items)
Include Subdomains	Boolean	1
Create Time	Number	535,682,980.29641
Expiry	Number	∞
HSTS Host	Boolean	1
www.logentries.com	Dictionary	(3 items)
webmail.onlime.ch	Dictionary	(4 items)

When displaying the output, the *create timestamp* is rendered correctly. The *expiration time*, however, sometimes uses an invalid timestamp value to represent infinity. For these cases, no expiration time is shown, but the raw data (or invalid value used for the timestamp) is still represented in the '*extra fields*' column.

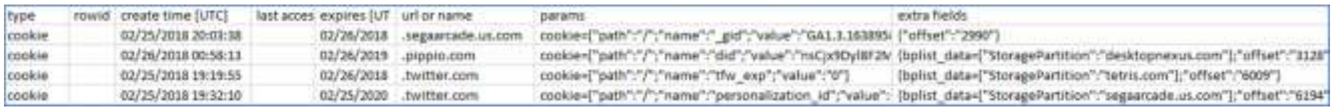


type	rowid	create time [UTC]	last acces	expires [UTC]	url or name	extra fie
Cookie HSTS (plist)		12/23/2017 00:49:40			{"dict_key_url":"com.apple.CFNetwork.defaultStorageSession/wiz.biz/"}	{"Create
Cookie HSTS (plist)		12/23/2017 00:49:40			{"dict_key_url":"com.apple.CFNetwork.defaultStorageSession/www.logentrie	{"Create
Cookie HSTS (plist)		12/23/2017 00:49:40			{"dict_key_url":"com.apple.CFNetwork.defaultStorageSession/webmail.onlim	{"Create
Cookie HSTS (plist)		12/23/2017 01:12:13		06/23/2018 13:12:13	{"dict_key_url":"com.apple.CFNetwork.defaultStorageSession/airbnb.com/"}	{"Create

extra fields
{"Create Time":"12/23/2017 00:49:40";"Expiry":"7ff0000000000000";"Include Subdomains":"true";"HSTS Host":"true"}
{"Create Time":"12/23/2017 00:49:40";"Expiry":"7ff0000000000000";"HSTS Host":"true"}
{"Create Time":"12/23/2017 00:49:40";"Expiry":"7ff0000000000000";"Include Subdomains":"true";"HSTS Host":"true"}
{"Create Time":"12/23/2017 01:12:13";"Expiry":"06/23/2018 13:12:13";"Include Subdomains":"true";"HSTS Host":"true"}

5.3 *binarycookies file*

The *binarycookies* output makes use of the '*params*' column. It uses this column to display details about the cookie. The '*extra fields*' column is used to display information about the cookie offset within the binary file, as well as any embedded *bplist* data contained in the cookie. From the empirical tests that were run on the embedded *bplist* data, it usually has information about the *StoragePartition*. In some cases, this embedded *bplist* data was not present. Below is a sample output from this cookie format.



type	rowid	create time [UTC]	last acces	expires [UT	url or name	params	extra fields
cookie		02/25/2018 20:03:38		02/26/2018	.segaarcade.us.com	cookie=["path":"/";"name":"_gid";"value":"GA1.3.163895	{"offset":"2990"}
cookie		02/26/2018 00:58:13		02/26/2019	.pippio.com	cookie=["path":"/";"name":"did";"value":"nsCjx9DylBF2V	{bplist_data=["StoragePartition":"desktopnexus.com"];"offset":"3128"
cookie		02/25/2018 19:19:55		02/26/2018	.twitter.com	cookie=["path":"/";"name":"tfw_exp";"value":"0"]	{bplist_data=["StoragePartition":"tetris.com"];"offset":"6009"}
cookie		02/25/2018 19:32:10		02/25/2020	.twitter.com	cookie=["path":"/";"name":"personalization_id";"value":	{bplist_data=["StoragePartition":"segaarcade.us.com"];"offset":"6194"

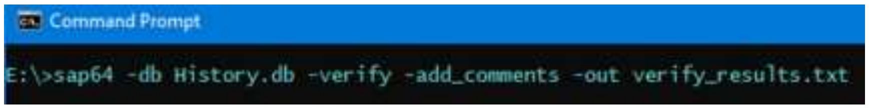
6 Verification and Validation (for SQLite files only)

All tools need to be 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 *Safari*, 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, ***sap*** 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 in order to 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 *Safari* 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 is 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.

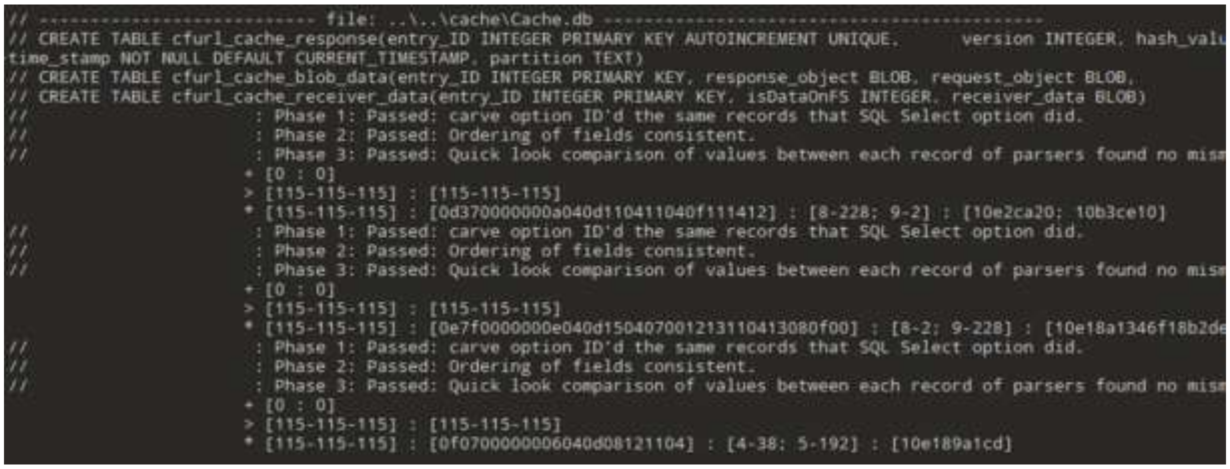
The nice thing about the way this option was implemented, is not only does it check the internal parsers against themselves, but it also outputs critical diagnostic data that can be used by TZWorks to help improve the tool. To ensure no personal information is outputted, the ***-verify*** option sanitizes the results to not contain 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.



```
E:\>sap64 -db History.db -verify -add_comments -out verify_results.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 of interest. 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.



```
// --------------------------- file: ..\..\cache\Cache.db ------------------------------------------
// CREATE TABLE cfurl_cache_response(entry_ID INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,      version INTEGER, hash_valu
time_stamp NOT NULL DEFAULT CURRENT_TIMESTAMP, partition TEXT)
// CREATE TABLE cfurl_cache_blob_data(entry_ID INTEGER PRIMARY KEY, response_object BLOB, request_object BLOB,
// CREATE TABLE cfurl_cache_receiver_data(entry_ID INTEGER PRIMARY KEY, isDataOnFS INTEGER, receiver_data BLOB)
//                        : Phase 1: Passed: carve option ID'd the same records that SQL Select option did.
//                        : Phase 2: Passed: Ordering of fields consistent.
//                        : Phase 3: Passed: Quick look comparison of values between each record of parsers found no mism
                         + [0 : 0]
                         > [115-115-115] : [115-115-115]
                         * [115-115-115] : [0d370000000a040d110411040f111412] : [8-228; 9-2] : [10e2ca20; 10b3ce10]
//                        : Phase 1: Passed: carve option ID'd the same records that SQL Select option did.
//                        : Phase 2: Passed: Ordering of fields consistent.
//                        : Phase 3: Passed: Quick look comparison of values between each record of parsers found no mism
                         + [0 : 0]
                         > [115-115-115] : [115-115-115]
                         * [115-115-115] : [0e7f0000000e040d15040700121311041308Of00] : [8-2; 9-228] : [10e18a1346f18b2de
//                        : Phase 1: Passed: carve option ID'd the same records that SQL Select option did.
//                        : Phase 2: Passed: Ordering of fields consistent.
//                        : Phase 3: Passed: Quick look comparison of values between each record of parsers found no mism
                         + [0 : 0]
                         > [115-115-115] : [115-115-115]
                         * [115-115-115] : [0f070000000006040d08121104] : [4-38; 5-192] : [10e189a1cd]
```

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 compares 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 *Safari* 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.

7 Use of the SQLite Library

For the purposes of ***sap***, 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.

8 CSV Field Names / Meaning

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

CSV Field	Definition
field	Cache version number
type	Type of data based on the table the record comes from. Example of types include: url, cookie, bookmark, favicon, download, etc
rowid	Internal parameter to the SQLite table record identifier. If a *plist* file is parsed, then this is the index or subkey of the record entry.
create time [UTC]	Date/Time the URL or item was created
last access [UTC]	Date/Time the URL or item was last visited or accessed
expires [UTC]	Date/Time the URL or item expires
url or name	URL or name of the item
params	Any HTTP parameters passed in with the URL (or can be used for other items if not a URL).
params translation	Translation of any parameter passed in (or can be used for other items if not some that require translation). Currently only used for SQLite files and not used for plist files
extra fields	Any fields not covered by the previous fields that are part of the record
data record source(s)	The source table and record offset within the database where this record was parsed (only applies to -carve and -parse_chunk parsing options)
file	Database file that was parsed

9 Limitations

This version of ***sap*** has a number of limitations. They are listed below.

- The tool is still a prototype in nature as this is the first version released. It still needs to be tested against various types of files, corrupted files, etc. to ensure the tool can perform consistently.
- The ***-split_session*** folder enumeration option relies on the *Safari* directory structure as well as the naming convention used by *Safari*. Therefore, if either of these things are changed by *Apple*

or *Safari* or if changed by a user, the parsing engine will have unpredictable results or no results at all.

10 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 from 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 the default or ***-csv*** options to remove any whitespace between the field value and the CSV separator.
-csv_separator	Only applies to ***-csv*** and ***-csvl2t*** options. Used in conjunction with the *-csv* option to change the CSV separator from the default comma to something else. Syntax is *-csv_separator "|"* to change the CSV separator to the pipe character. To use the tab as a separator, one can use the *-csv_separator "tab"* OR *-csv_separator "\t"* options.
-dateformat	Output the date using the specified format. Default behavior is *-dateformat "yyyy-mm-dd"*. Using this option allows one to adjust the format to

	mm/dd/yy, dd/mm/yy, etc. The restriction with this option is the forward slash (/) or dash (-) symbol needs to separate month, day and year and the month is in digit (1-12) form versus abbreviated name form.
-timeformat	Output the time using the specified format. Default behavior is ***-timeformat "hh:mm:ss.xxx".*** One can adjust the format to microseconds, via ***"hh:mm:ss.xxxxxx"*** or nanoseconds, via ***"hh:mm:ss.xxxxxxxxx"***, or no fractional seconds, via ***"hh:mm:ss"***. The 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.
-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 at unused space to see if any records are present. Not required with the ***-parse_chunk*** option. Use this in conjunction with ***-carve*** or default option to look for discarded 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).
-blob	(sub option for ***-parse_chunk***). Experimental option. The ***-parse_chunk*** option will try to determine the type of data present in the SQLite file based on its filename. Use this option, as a secondary approach, for those cases when the filename is unrelated to the file type for the artifact. For these cases, the parser will try all the signatures to try to extract records. (*Warning*. This option can create many false positives).
-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.
-quiet	Show no progress during the parsing operation.
-split_sessions	Split the *Safari* sessions into separate files.
-show_all_data	Do not on truncate long data runs

-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.

11 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.

12 References

- *SQLite library statically linked into tool [Amalgamation of many separate C source files from SQLite version 3.32.3].*
- *SQLite documentation [http://www.sqlite.org].*
- *DB Browser for SQLite [http://sqlitebrowser.org/]*
- *https://github.com/libyal/dtformats/blob/main/documentation/Safari%20Cookies.asciidoc*
- *https://stackoverflow.com/questions/7545885/safari-5-1-cookie-format-specs*
- *http://www.securitylearn.net/2012/10/27/cookies-binarycookies-reader/*
- *https://opensource.apple.com/source/CF/CF-550/CFBinaryPList.c*
- *https://medium.com/@karaiskc/understanding-apples-binary-property-list-format-281e6da00dbd*
- *http://newosxbook.com/bonus/bplist.pdf*