Browser Artifacts

Table of Contents

Introduction

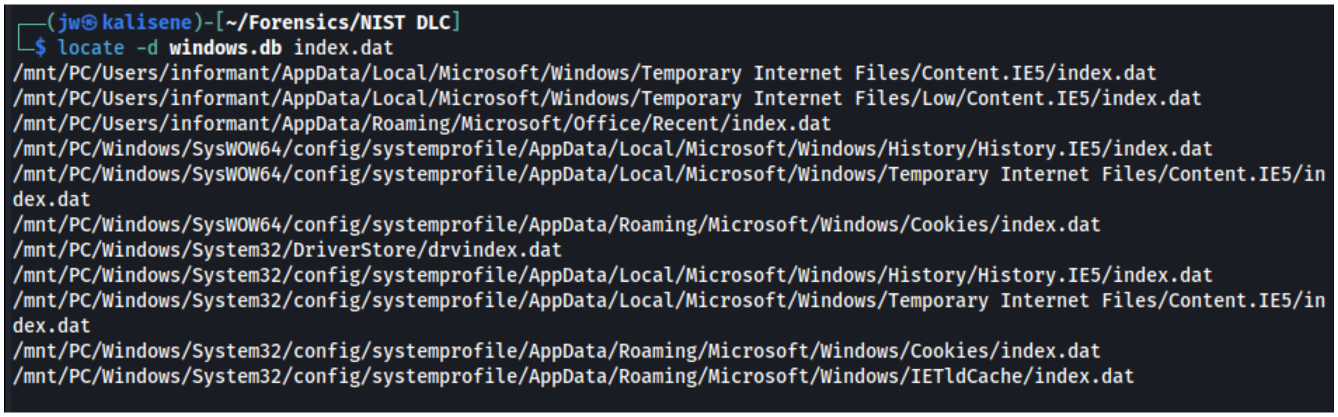
Since the Date Leakage Case only includes IE and Chrome, those are the only browsers I'll go over in this walkthrough. Browsers have very valuable, detailed information if they are configured properly. The information they do provide, however, can be a little overwhelming. Extracting the information is straightforward, but gathering proper information still requires manual analysis and problem solving.

Internet Explorer

Reading:

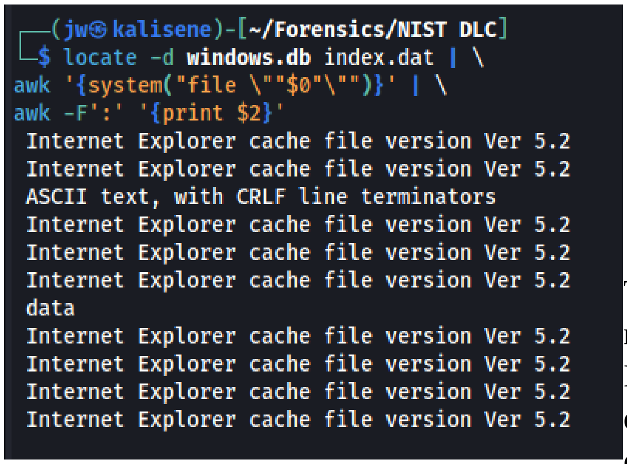
- https://forensics.wiki/internet_explorer_history_file_format/
- https://forensics.wiki/libmsiecf/
- https://forensics.wiki/internet_explorer/#msie-10
- https://forensics.wiki/extensible_storage_engine_%28ese%29_database_file_%28edb%29_format/
- https://github.com/libyal/libesedb/

Different versions of Internet Explorer store history in different locations. Earlier versions store it in a file called **index.dat**, so let's try searching for those files.



```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC]
└─$ locate -d windows.db index.dat
/mnt/PC/Users/informant/AppData/Local/Microsoft/Windows/Temporary Internet Files/Content.IE5/index.dat
/mnt/PC/Users/informant/AppData/Local/Microsoft/Windows/Temporary Internet Files/Low/Content.IE5/index.dat
/mnt/PC/Users/informant/AppData/Roaming/Microsoft/Office/Recent/index.dat
/mnt/PC/Windows/SysWOW64/config/systemprofile/AppData/Local/Microsoft/Windows/History/History.IE5/index.dat
/mnt/PC/Windows/SysWOW64/config/systemprofile/AppData/Local/Microsoft/Windows/Temporary Internet Files/Content.IE5/in
dex.dat
/mnt/PC/Windows/SysWOW64/config/systemprofile/AppData/Roaming/Microsoft/Windows/Cookies/index.dat
/mnt/PC/Windows/System32/DriverStore/drvindex.dat
/mnt/PC/Windows/System32/config/systemprofile/AppData/Local/Microsoft/Windows/History/History.IE5/index.dat
/mnt/PC/Windows/System32/config/systemprofile/AppData/Local/Microsoft/Windows/Temporary Internet Files/Content.IE5/in
dex.dat
/mnt/PC/Windows/System32/config/systemprofile/AppData/Roaming/Microsoft/Windows/Cookies/index.dat
/mnt/PC/Windows/System32/config/systemprofile/AppData/Roaming/Microsoft/Windows/IETldCache/index.dat
```

"index.dat" is not a name unique to IE, however. Confirm the contents with the **file** command:



```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC]
└─$ locate -d windows.db index.dat | \
awk '{system("file \""$0"\"")}' | \
awk -F':' '{print $2}'
 Internet Explorer cache file version Ver 5.2
 Internet Explorer cache file version Ver 5.2
 ASCII text, with CRLF line terminators
 Internet Explorer cache file version Ver 5.2
 Internet Explorer cache file version Ver 5.2
 Internet Explorer cache file version Ver 5.2
 data
 Internet Explorer cache file version Ver 5.2
 Internet Explorer cache file version Ver 5.2
 Internet Explorer cache file version Ver 5.2
 Internet Explorer cache file version Ver 5.2
```

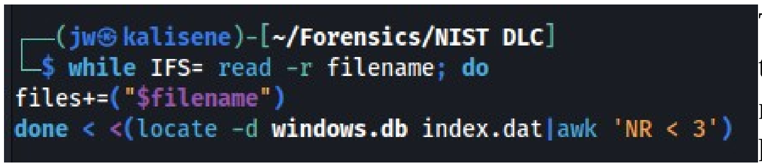
1. **Locate** all files named 'index.dat'
2. Enclose the output in double quotes, then run **file** on them.
3. Print the output from **file**, for improved readability.

That MS Office file might be interesting later, but for now, I want to look some of these IE cache files. Individual users' history is stored in their home directory, so I'm only going to parse the first two options from that output.

For the actual parsing, I’m using a library called **libmsiecf** (Library for Microsoft Internet Explorer Cache Files). It comes with two utilities:

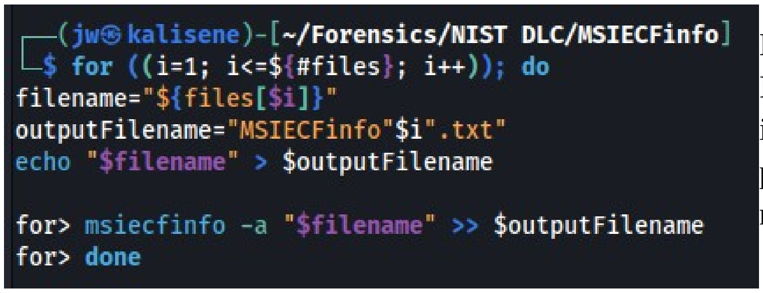
- **msiecfinfo**: Provides quick info about IE cache files.
- **Msiecfexport**: Exports IE cache files into a readable format.

We’re going to run these commands on the first two IE cache files. I want to run them in a loop, primarily to get the practice of using more powerful constructs on the command line.



```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC]
└─$ while IFS= read -r filename; do
files+=("$filename")
done < <(locate -d windows.db index.dat|awk 'NR < 3')
```

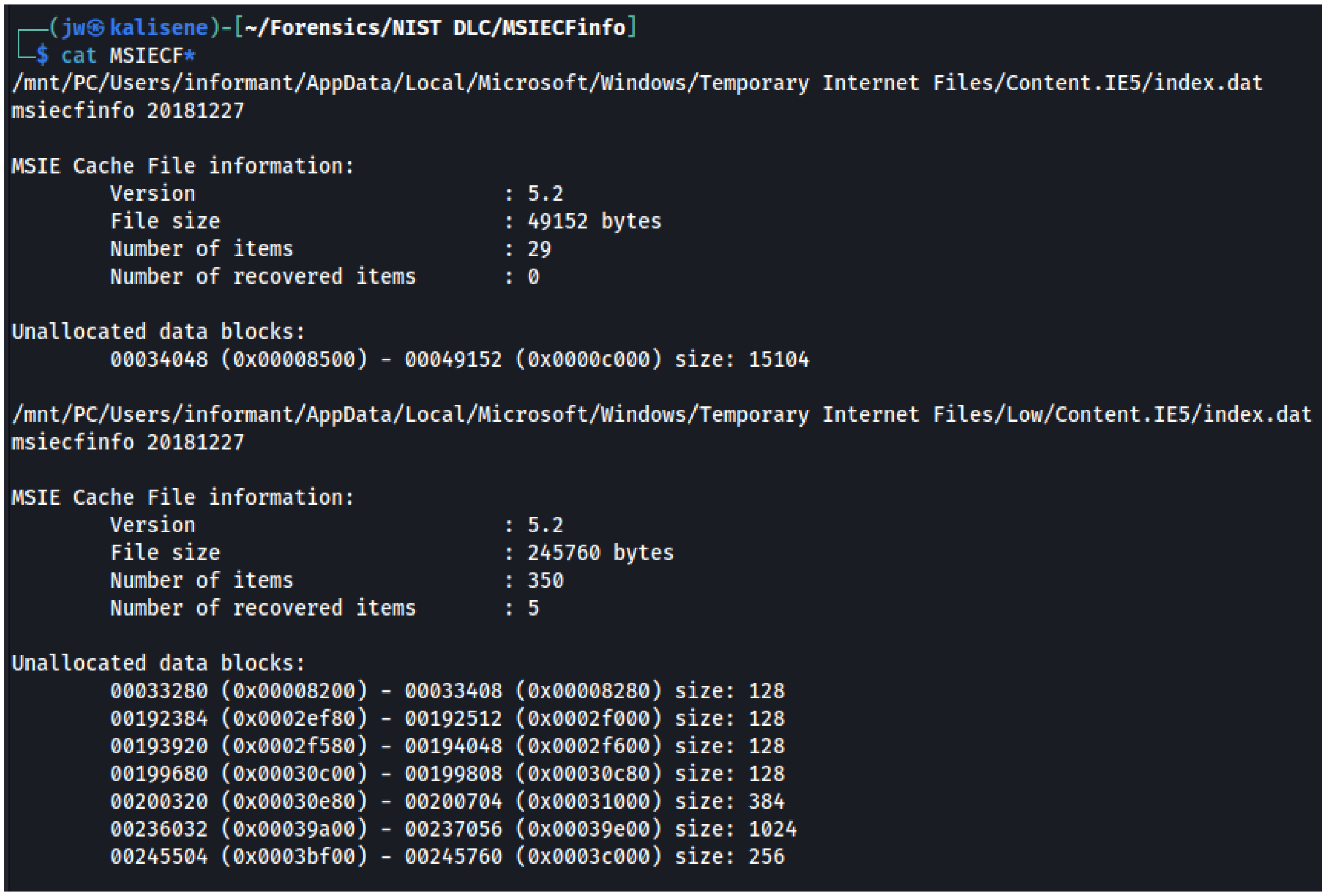
This is building an array called **files**. Note the quotes around **filename**. This is necessary because some of the file paths have spaces in them, so they won’t be stored correctly. The **awk** command simply grabs the first two files.



```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/MSIECFinfo]
└─$ for ((i=1; i<=${#files}; i++)); do
filename="${files[$i]}"
outputFilename="MSIECFinfo"$i".txt"
echo "$filename" > $outputFilename

for> msiecfinfo -a "$filename" >> $outputFilename
for> done
```

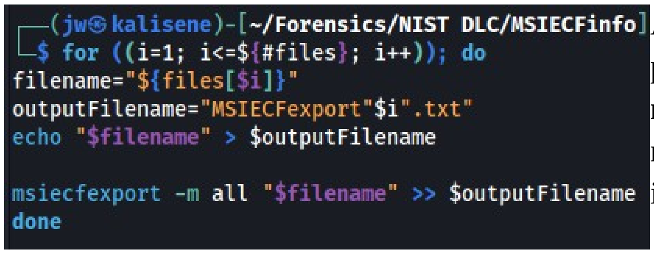
Now that I have a list of files to work with, I built a loop. Note that in Zsh, arrays are indexed at 1, instead of 0 with most other programming languages. It’s a strange nuance of Zsh.

Now let’s look at the output from **msiecfinfo:**



```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/MSIECFinfo]
└─$ cat MSIECF*
/mnt/PC/Users/informant/AppData/Local/Microsoft/Windows/Temporary Internet Files/Content.IE5/index.dat
msiecfinfo 20181227

MSIE Cache File information:
        Version                         : 5.2
        File size                       : 49152 bytes
        Number of items                 : 29
        Number of recovered items       : 0

Unallocated data blocks:
        00034048 (0x00008500) - 00049152 (0x0000c000) size: 15104

/mnt/PC/Users/informant/AppData/Local/Microsoft/Windows/Temporary Internet Files/Low/Content.IE5/index.dat
msiecfinfo 20181227

MSIE Cache File information:
        Version                         : 5.2
        File size                       : 245760 bytes
        Number of items                 : 350
        Number of recovered items       : 5

Unallocated data blocks:
        00033280 (0x00008200) - 00033408 (0x00008280) size: 128
        00192384 (0x0002ef80) - 00192512 (0x0002f000) size: 128
        00193920 (0x0002f580) - 00194048 (0x0002f600) size: 128
        00199680 (0x00030c00) - 00199808 (0x00030c80) size: 128
        00200320 (0x00030e80) - 00200704 (0x00031000) size: 384
        00236032 (0x00039a00) - 00237056 (0x00039e00) size: 1024
        00245504 (0x0003bf00) - 00245760 (0x0003c000) size: 256
```

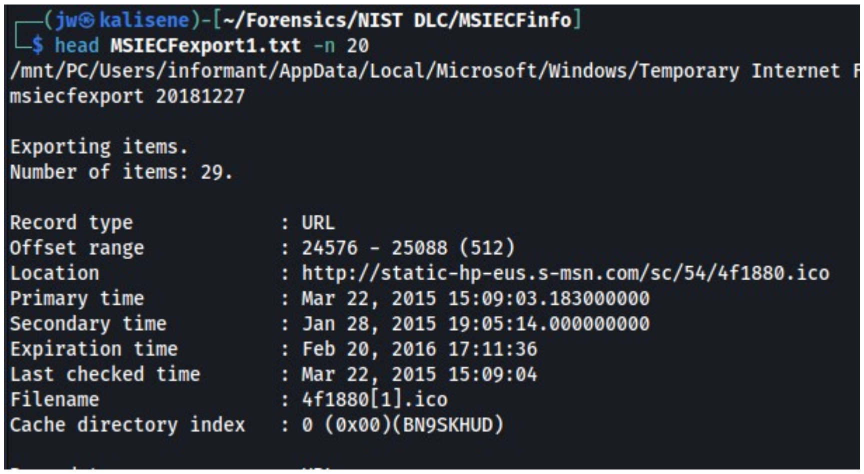
The most important information to glean here is the **number of items/recovered items**. This just tells you roughly how much you have to parse afterwards—as I said, this just provides quick info on the cache files. Let’s try exporting them.



```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/MSIECFinfo]
└─$ for ((i=1; i<=${#files}; i++)); do
filename="${files[$i]}"
outputFilename="MSIECFexport"$i".txt"
echo "$filename" > $outputFilename

msiecfexport -m all "$filename" >> $outputFilename
done
```

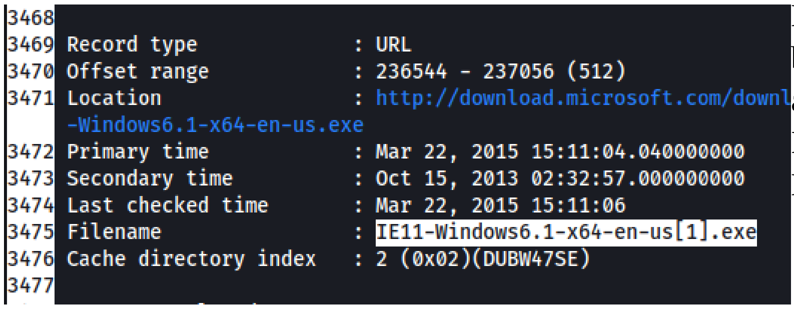
All I had to do was change “info” to “export” in my previous loop command, and edit the call to **msiecfexport**. The **-m all** options exports items and recovered items, but you can export them separately if you want.

Finally we can look at the subject's browsing history. The exported information gives us a time, a place and sometimes a filename.



```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/MSIECFinfo]
└─$ head MSIECFexport1.txt -n 20
/mnt/PC/Users/informant/AppData/Local/Microsoft/Windows/Temporary Internet F
msiecfexport 20181227

Exporting items.
Number of items: 29.

Record type             : URL
Offset range            : 24576 - 25088 (512)
Location                : http://static-hp-eus.s-msn.com/sc/54/4f1880.ico
Primary time            : Mar 22, 2015 15:09:03.183000000
Secondary time          : Jan 28, 2015 19:05:14.000000000
Expiration time         : Feb 20, 2016 17:11:36
Last checked time       : Mar 22, 2015 15:09:04
Filename                : 4f1880[1].ico
Cache directory index   : 0 (0x00)(BN9SKHUD)
```

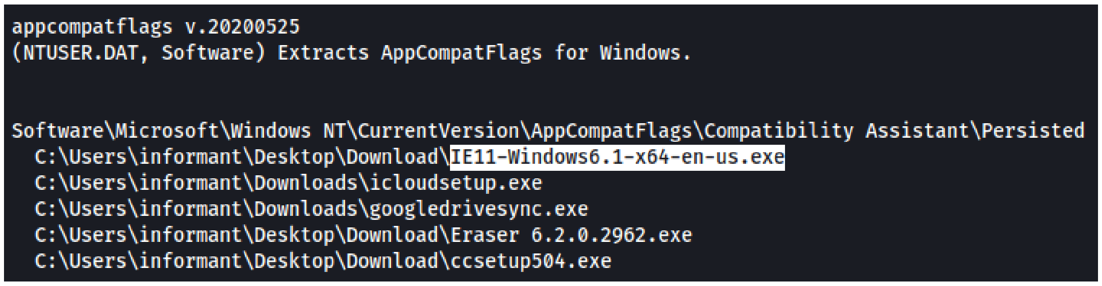
This is where these tools stop, however. It is now up to you, the forensic analyst, to parse the remaining information. Not all the URLs and files will be relevant to the investigation, especially given that some files are simply assets from websites. I find these tools easy to use, but they leave much to be done, which isn't necessarily a problem. We can still use them to understand the data contained within the source files. Regardless, I leave the parsing of the output to you.



```

Record type             : URL
Offset range            : 236544 - 237056 (512)
Location                : http://download.microsoft.com/downl
-Windows6.1-x64-en-us.exe
Primary time            : Mar 22, 2015 15:11:04.040000000
Secondary time          : Oct 15, 2013 02:32:57.000000000
Last checked time       : Mar 22, 2015 15:11:06
Filename                : IE11-Windows6.1-x64-en-us[1].exe
Cache directory index   : 2 (0x02)(DUBW47SE)

```

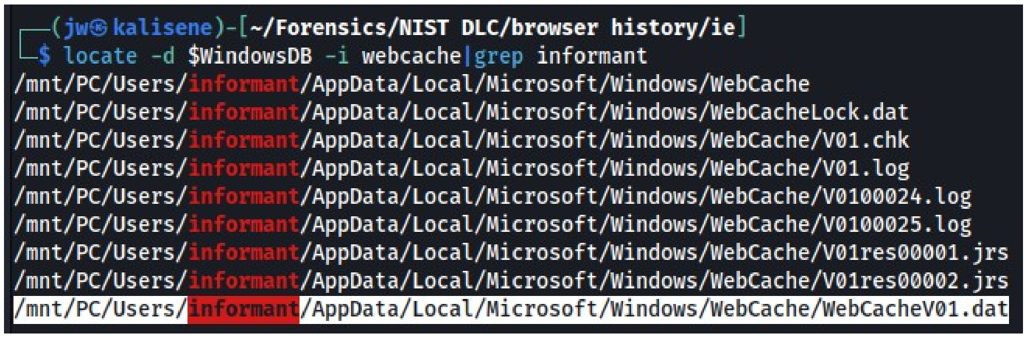
Moving on, the recovered items are at the bottom of the output file, and there's actually something very interesting there. It looks like a download link for Internet Explorer version 11.

And if you used RegRipper with the **appcompatflags** plugin, you'd find the same file, so we can be sure the subject downloaded Internet Explorer 11.



```
appcompatflags v.20200525
(NTUSER.DAT, Software) Extracts AppCompatFlags for Windows.


Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Compatibility Assistant\Persisted
  C:\Users\informant\Desktop\Download\IE11-Windows6.1-x64-en-us.exe
  C:\Users\informant\Downloads\icloudsetup.exe
  C:\Users\informant\Downloads\googledrivesync.exe
  C:\Users\informant\Desktop\Download\Eraser 6.2.0.2962.exe
  C:\Users\informant\Desktop\Download\ccsetup504.exe
```

A cursory web search shows that later versions of Internet Explorer use a different file format for history: the **extensible storage engine database file** (EDB) format in a file called **WebCacheV01.dat**. If we search for this file:

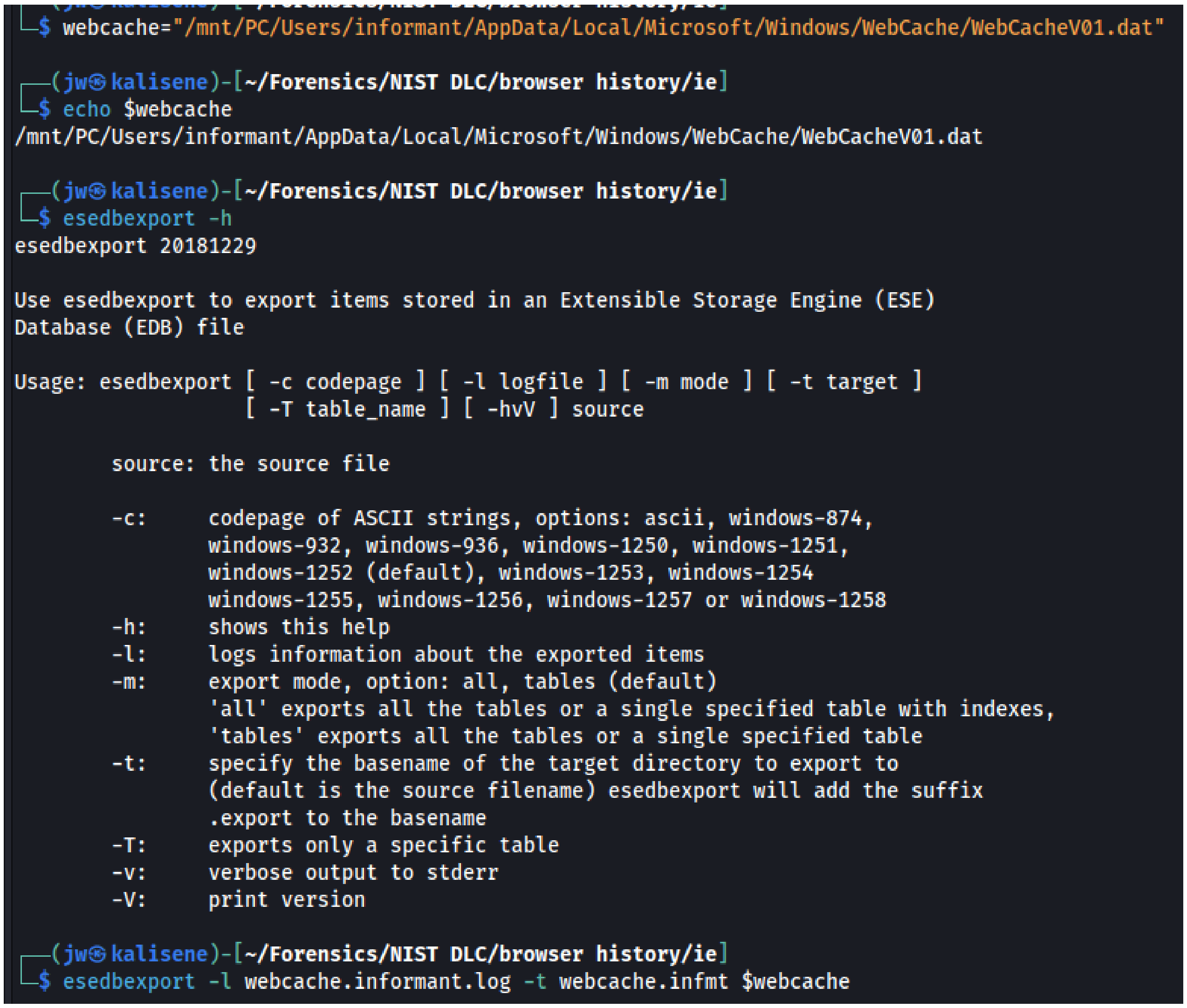


```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/browser history/ie]
└─$ locate -d $WindowsDB -i webcache|grep informant
/mnt/PC/Users/informant/AppData/Local/Microsoft/Windows/WebCache
/mnt/PC/Users/informant/AppData/Local/Microsoft/Windows/WebCacheLock.dat
/mnt/PC/Users/informant/AppData/Local/Microsoft/Windows/WebCache/V01.chk
/mnt/PC/Users/informant/AppData/Local/Microsoft/Windows/WebCache/V01.log
/mnt/PC/Users/informant/AppData/Local/Microsoft/Windows/WebCache/V0100024.log
/mnt/PC/Users/informant/AppData/Local/Microsoft/Windows/WebCache/V0100025.log
/mnt/PC/Users/informant/AppData/Local/Microsoft/Windows/WebCache/V01res00001.jrs
/mnt/PC/Users/informant/AppData/Local/Microsoft/Windows/WebCache/V01res00002.jrs
/mnt/PC/Users/informant/AppData/Local/Microsoft/Windows/WebCache/WebCacheV01.dat
```

Now we need some tools that can parse EDB files. Luckily, we have **libesedb**:

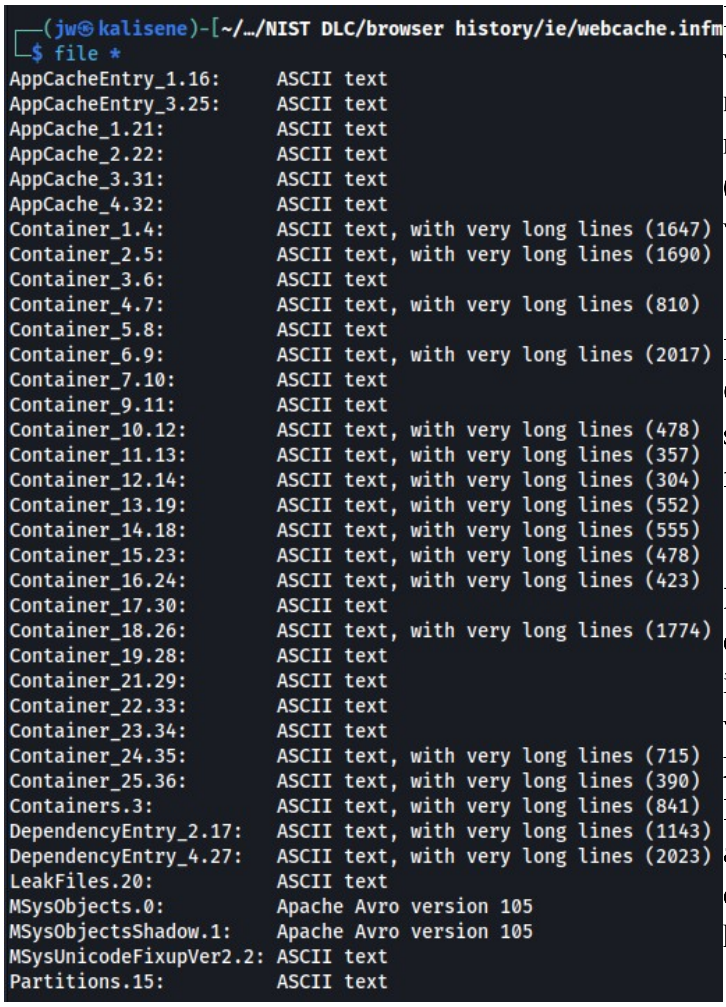
- **esedbexport**: Exports the EDB file into a readable format.
- **esedbinfo**: Prints information about the EDB file.

The steps to use these tools are the same as libmsiecf.



```
└─$ webcache="/mnt/PC/Users/informant/AppData/Local/Microsoft/Windows/WebCache/WebCacheV01.dat"

┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/browser history/ie]
└─$ echo $webcache
/mnt/PC/Users/informant/AppData/Local/Microsoft/Windows/WebCache/WebCacheV01.dat

┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/browser history/ie]
└─$ esedbexport -h
esedbexport 20181229

Use esedbexport to export items stored in an Extensible Storage Engine (ESE)
Database (EDB) file

Usage: esedbexport [ -c codepage ] [ -l logfile ] [ -m mode ] [ -t target ]
                   [ -T table_name ] [ -hvV ] source

        source: the source file

        -c:     codepage of ASCII strings, options: ascii, windows-874,
                windows-932, windows-936, windows-1250, windows-1251,
                windows-1252 (default), windows-1253, windows-1254
                windows-1255, windows-1256, windows-1257 or windows-1258
        -h:     shows this help
        -l:     logs information about the exported items
        -m:     export mode, option: all, tables (default)
                'all' exports all the tables or a single specified table with indexes,
                'tables' exports all the tables or a single specified table
        -t:     specify the basename of the target directory to export to
                (default is the source filename) esedbexport will add the suffix
                .export to the basename
        -T:     exports only a specific table
        -v:     verbose output to stderr
        -V:     print version

┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/browser history/ie]
└─$ esedbexport -l webcache.informant.log -t webcache.infmt $webcache
```

Figure 1: I made a local variable for the webcache file, so referencing it was easier.

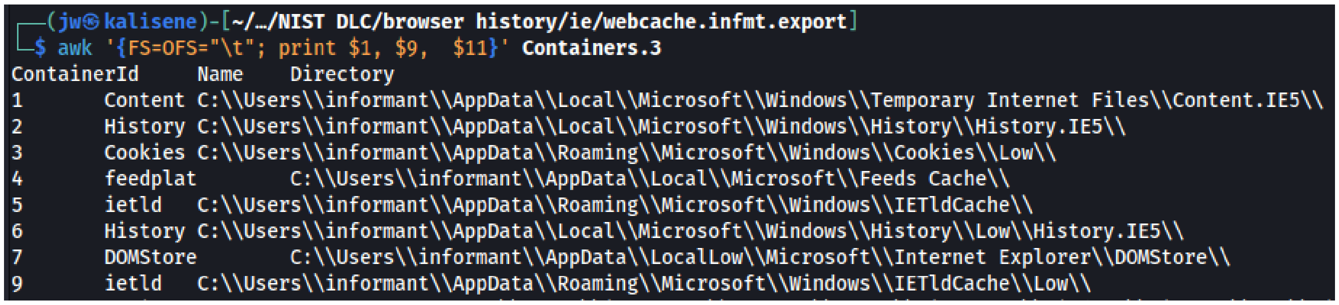


```
┌──(jw㉿kalisene)-[~/…/NIST DLC/browser history/ie/webcache.infm
└─$ file *
AppCacheEntry_1.16:     ASCII text
AppCacheEntry_3.25:     ASCII text
AppCache_1.21:          ASCII text
AppCache_2.22:          ASCII text
AppCache_3.31:          ASCII text
AppCache_4.32:          ASCII text
Container_1.4:          ASCII text, with very long lines (1647)
Container_2.5:          ASCII text, with very long lines (1690)
Container_3.6:          ASCII text
Container_4.7:          ASCII text, with very long lines (810)
Container_5.8:          ASCII text
Container_6.9:          ASCII text, with very long lines (2017)
Container_7.10:         ASCII text
Container_9.11:         ASCII text
Container_10.12:        ASCII text, with very long lines (478)
Container_11.13:        ASCII text, with very long lines (357)
Container_12.14:        ASCII text, with very long lines (304)
Container_13.19:        ASCII text, with very long lines (552)
Container_14.18:        ASCII text, with very long lines (555)
Container_15.23:        ASCII text, with very long lines (478)
Container_16.24:        ASCII text, with very long lines (423)
Container_17.30:        ASCII text
Container_18.26:        ASCII text, with very long lines (1774)
Container_19.28:        ASCII text
Container_21.29:        ASCII text
Container_22.33:        ASCII text
Container_23.34:        ASCII text
Container_24.35:        ASCII text, with very long lines (715)
Container_25.36:        ASCII text, with very long lines (390)
Containers.3:           ASCII text, with very long lines (841)
DependencyEntry_2.17:   ASCII text, with very long lines (1143)
DependencyEntry_4.27:   ASCII text, with very long lines (2023)
LeakFiles.20:           ASCII text
MSysObjects.0:          Apache Avro version 105
MSysObjectsShadow.1:    Apache Avro version 105
MSysUnicodeFixupVer2.2: ASCII text
Partitions.15:          ASCII text
```

It exports a lot of files. Unfortunately, this is where **esedbexport** stops. I haven't found many resources for this tool either, which means we have to parse the output manually (barring a high powered framework, which I will cover in a later tutorial).

Looking at the list, one file stands out: **Containers.3**. Notice every other file that starts with 'Container' has an underscore followed by 2 numbers.

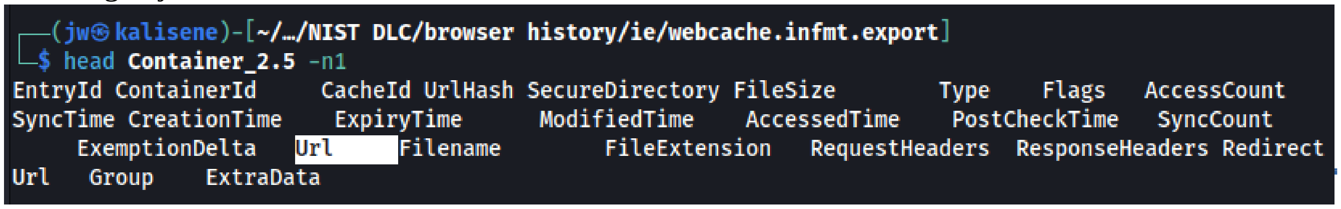
Furthermore, the header on Containers.3 is different from the other Container files, and includes a column for **Container ID**, with values from 1 to 25. There is also a column for **Name**, which implies what each file contains. I believe Containers.3 contains metadata about all the other exported files. Looking at the different names, we find some potential history files in containers 2 and 6:



```
┌──(jw㉿kalisene)-[~/…/NIST DLC/browser history/ie/webcache.infmt.export]
└─$ awk '{FS=OFS="\t"; print $1, $9,  $11}' Containers.3
ContainerId     Name    Directory
1       Content C:\\Users\\informant\\AppData\\Local\\Microsoft\\Windows\\Temporary Internet Files\\Content.IE5\\
2       History C:\\Users\\informant\\AppData\\Local\\Microsoft\\Windows\\History\\History.IE5\\
3       Cookies C:\\Users\\informant\\AppData\\Roaming\\Microsoft\\Windows\\Cookies\\Low\\
4       feedplat        C:\\Users\\informant\\AppData\\Local\\Microsoft\\Feeds Cache\\
5       ietld   C:\\Users\\informant\\AppData\\Roaming\\Microsoft\\Windows\\IETldCache\\
6       History C:\\Users\\informant\\AppData\\Local\\Microsoft\\Windows\\History\\Low\\History.IE5\\
7       DOMStore        C:\\Users\\informant\\AppData\\LocalLow\\Microsoft\\Internet Explorer\\DOMStore\\
9       ietld   C:\\Users\\informant\\AppData\\Roaming\\Microsoft\\Windows\\IETldCache\\Low\\
```

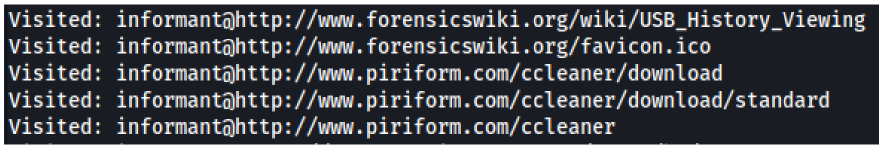
Disclaimer: I haven't confirmed my findings about esedbexport. I'm just using context clues to figure out where to go. Regardless, let's take a look at the first history file, Container_2.5.

Looking at just the header, we see a record for URL:



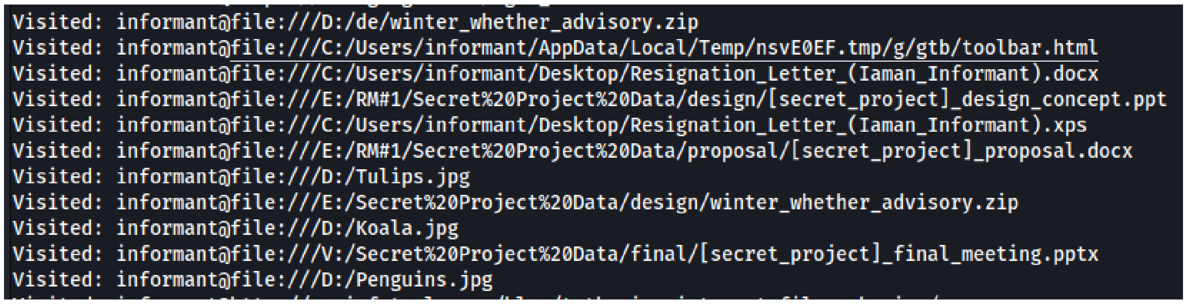
```
┌──(jw㉿kalisene)-[~/…/NIST DLC/browser history/ie/webcache.infmt.export]
└─$ head Container_2.5 -n1
EntryId ContainerId     CacheId UrlHash SecureDirectory FileSize        Type    Flags   AccessCount
SyncTime CreationTime     ExpiryTime       ModifiedTime     AccessedTime     PostCheckTime    SyncCount
     ExemptionDelta     Url      Filename         FileExtension    RequestHeaders  ResponseHeaders Redirect
Url    Group    ExtraData
```

Looking at the URL column, we find lots of valuable information:



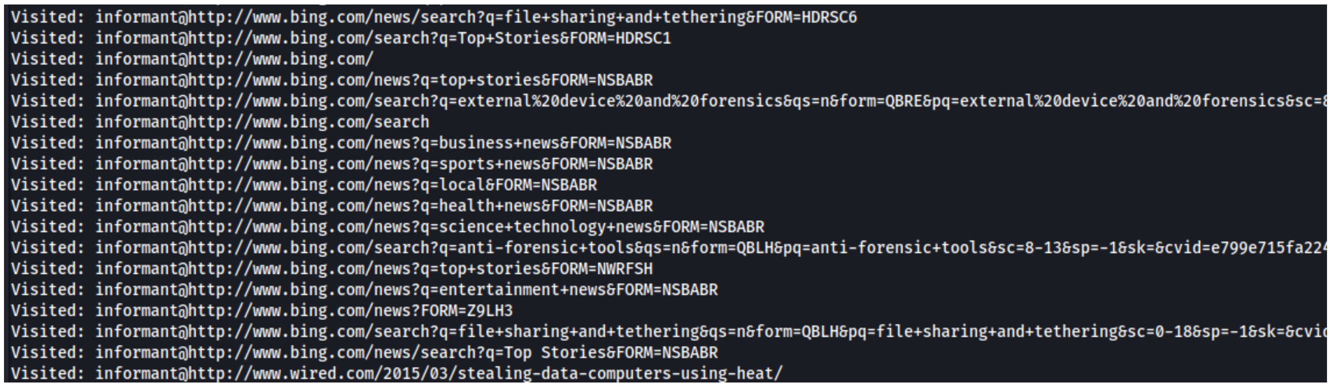
```
Visited: informant@http://www.forensicswiki.org/wiki/USB_History_Viewing
Visited: informant@http://www.forensicswiki.org/favicon.ico
Visited: informant@http://www.piriform.com/ccleaner/download
Visited: informant@http://www.piriform.com/ccleaner/download/standard
Visited: informant@http://www.piriform.com/ccleaner
```

Figure 2: A download link for CCleaner, which we identified earlier.



```
Visited: informant@file:///D:/de/winter_whether_advisory.zip
Visited: informant@file:///C:/Users/informant/AppData/Local/Temp/nsvE0EF.tmp/g/gtb/toolbar.html
Visited: informant@file:///C:/Users/informant/Desktop/Resignation_Letter_(Iaman_Informant).docx
Visited: informant@file:///E:/RM#1/Secret%20Project%20Data/design/[secret_project]_design_concept.ppt
Visited: informant@file:///C:/Users/informant/Desktop/Resignation_Letter_(Iaman_Informant).xps
Visited: informant@file:///E:/RM#1/Secret%20Project%20Data/proposal/[secret_project]_proposal.docx
Visited: informant@file:///D:/Tulips.jpg
Visited: informant@file:///E:/Secret%20Project%20Data/design/winter_whether_advisory.zip
Visited: informant@file:///D:/Koala.jpg
Visited: informant@file:///V:/Secret%20Project%20Data/final/[secret_project]_final_meeting.pptx
Visited: informant@file:///D:/Penguins.jpg
```

Figure 3: Local files, which I'll cover in another tutorial.



```
Visited: informant@http://www.bing.com/news/search?q=file+sharing+and+tethering&FORM=HDRSC6
Visited: informant@http://www.bing.com/search?q=Top+Stories&FORM=HDRSC1
Visited: informant@http://www.bing.com/
Visited: informant@http://www.bing.com/news?q=top+stories&FORM=NSBABR
Visited: informant@http://www.bing.com/search?q=external%20device%20and%20forensics&qs=n&form=QBRE&pq=external%20device%20and%20forensics&sc=8
Visited: informant@http://www.bing.com/search
Visited: informant@http://www.bing.com/news?q=business+news&FORM=NSBABR
Visited: informant@http://www.bing.com/news?q=sports+news&FORM=NSBABR
Visited: informant@http://www.bing.com/news?q=local&FORM=NSBABR
Visited: informant@http://www.bing.com/news?q=health+news&FORM=NSBABR
Visited: informant@http://www.bing.com/news?q=science+technology+news&FORM=NSBABR
Visited: informant@http://www.bing.com/search?q=anti-forensic+tools&qs=n&form=QBLH&pq=anti-forensic+tools&sc=8-13&sp=-1&sk=&cvid=e799e715fa224
Visited: informant@http://www.bing.com/news?q=top+stories&FORM=NWRFSH
Visited: informant@http://www.bing.com/news?q=entertainment+news&FORM=NSBABR
Visited: informant@http://www.bing.com/news?FORM=Z9LH3
Visited: informant@http://www.bing.com/search?q=file+sharing+and+tethering&qs=n&form=QBLH&pq=file+sharing+and+tethering&sc=0-18&sp=-1&sk=&cvic
Visited: informant@http://www.bing.com/news/search?q=Top Stories&FORM=NSBABR
Visited: informant@http://www.wired.com/2015/03/stealing-data-computers-using-heat/
```

Figure 4: Suspicious search terms. Hard to read as is but can be formatted with a little work.

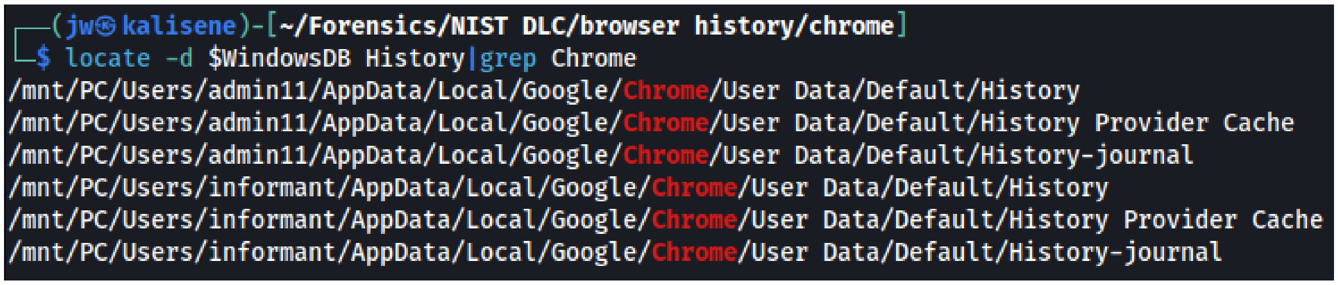
As the forensic analyst, you’ll need to go through this and find the most significant artifacts. You’ll find similar artifacts in the container labelled 6.

Google Chrome

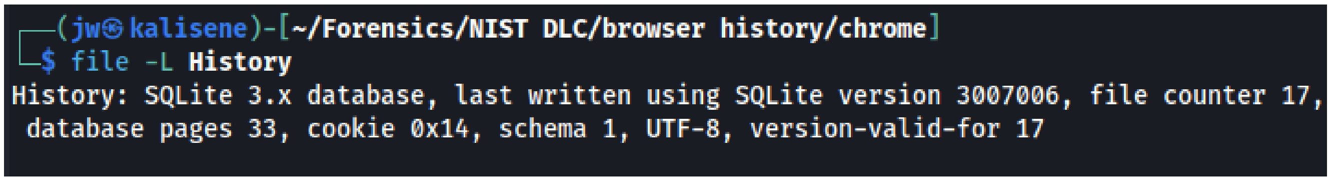
Reading:

- https://forensics.wiki/google_chrome/
- https://www.epochconverter.com/webkit

Chrome stores its history in a file neatly named **History**:



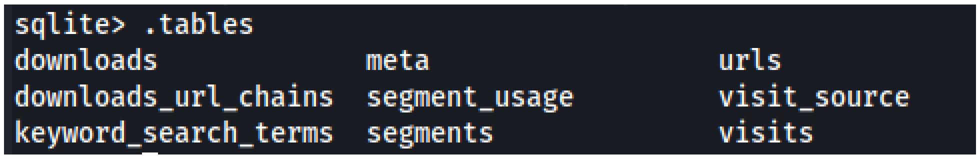
```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/browser history/chrome]
└─$ locate -d $WindowsDB History|grep Chrome
/mnt/PC/Users/admin11/AppData/Local/Google/Chrome/User Data/Default/History
/mnt/PC/Users/admin11/AppData/Local/Google/Chrome/User Data/Default/History Provider Cache
/mnt/PC/Users/admin11/AppData/Local/Google/Chrome/User Data/Default/History-journal
/mnt/PC/Users/informant/AppData/Local/Google/Chrome/User Data/Default/History
/mnt/PC/Users/informant/AppData/Local/Google/Chrome/User Data/Default/History Provider Cache
/mnt/PC/Users/informant/AppData/Local/Google/Chrome/User Data/Default/History-journal
```



```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/browser history/chrome]
└─$ file -L History
History: SQLite 3.x database, last written using SQLite version 3007006, file counter 17,
 database pages 33, cookie 0x14, schema 1, UTF-8, version-valid-for 17
```

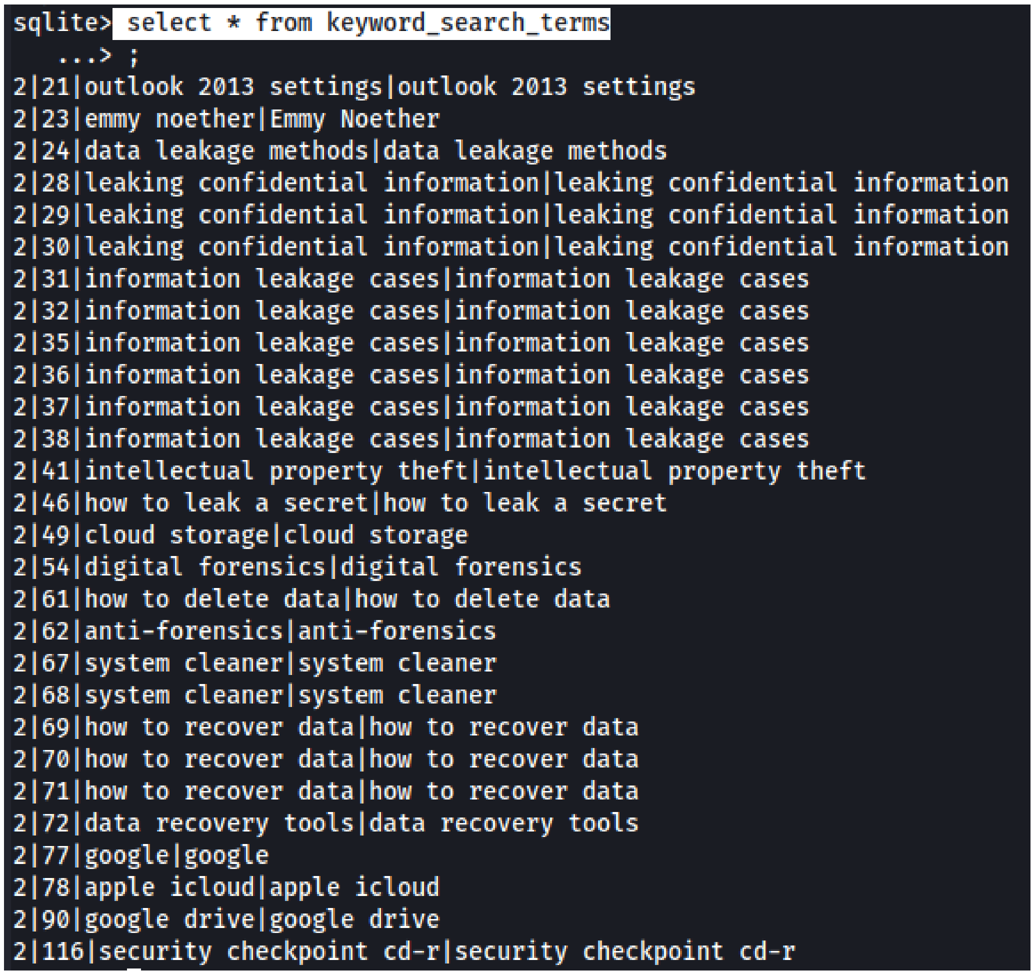
Figure 5: I symlinked it to my working directory for ease of access.

Kali comes with a SQLite3 command line utility to browse SQLite databases. If you prefer a GUI, it also comes with a GUI option, but I like working on the command line. Looking at the tables in the database, they are labelled very clearly:



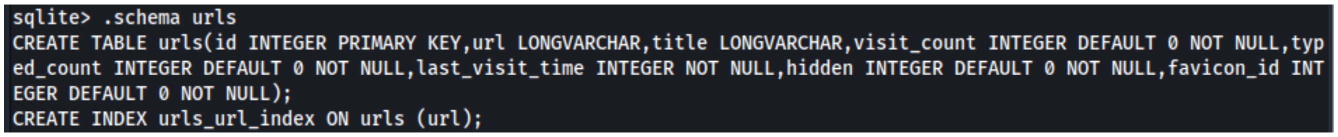
```
sqlite> .tables
downloads             meta                  urls
downloads_url_chains  segment_usage         visit_source
keyword_search_terms  segments              visits
```

Looking at the *keyword_search_terms* table, there are some very obvious searches. Note that these entries are what the user *actually typed* in the search bar.

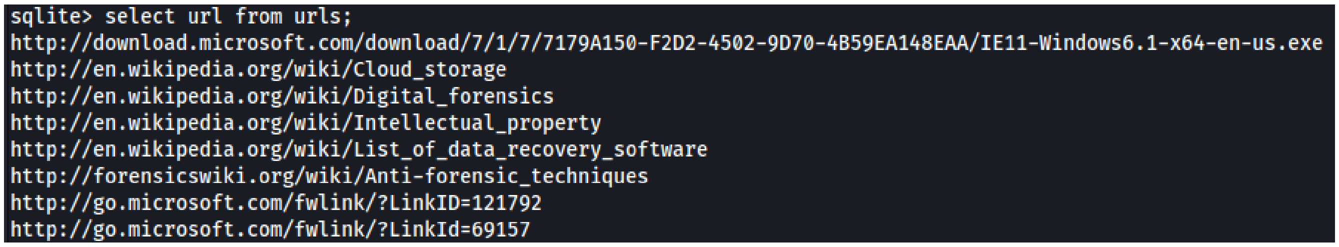


```
sqlite> select * from keyword_search_terms
   ...> ;
2|21|outlook 2013 settings|outlook 2013 settings
2|23|emmy noether|Emmy Noether
2|24|data leakage methods|data leakage methods
2|28|leaking confidential information|leaking confidential information
2|29|leaking confidential information|leaking confidential information
2|30|leaking confidential information|leaking confidential information
2|31|information leakage cases|information leakage cases
2|32|information leakage cases|information leakage cases
2|35|information leakage cases|information leakage cases
2|36|information leakage cases|information leakage cases
2|37|information leakage cases|information leakage cases
2|38|information leakage cases|information leakage cases
2|41|intellectual property theft|intellectual property theft
2|46|how to leak a secret|how to leak a secret
2|49|cloud storage|cloud storage
2|54|digital forensics|digital forensics
2|61|how to delete data|how to delete data
2|62|anti-forensics|anti-forensics
2|67|system cleaner|system cleaner
2|68|system cleaner|system cleaner
2|69|how to recover data|how to recover data
2|70|how to recover data|how to recover data
2|71|how to recover data|how to recover data
2|72|data recovery tools|data recovery tools
2|77|google|google
2|78|apple icloud|apple icloud
2|90|google drive|google drive
2|116|security checkpoint cd-r|security checkpoint cd-r
```

The URLs table is a bit more complicated. Of the data types it stores, *url*, and *last_visit_time* seem to be the most significant.

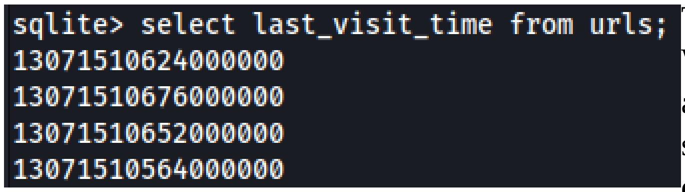


```
sqlite> .schema urls
CREATE TABLE urls(id INTEGER PRIMARY KEY,url LONGVARCHAR,title LONGVARCHAR,visit_count INTEGER DEFAULT 0 NOT NULL,typ
ed_count INTEGER DEFAULT 0 NOT NULL,last_visit_time INTEGER NOT NULL,hidden INTEGER DEFAULT 0 NOT NULL,favicon_id INT
EGER DEFAULT 0 NOT NULL);
CREATE INDEX urls_url_index ON urls (url);
```



```
sqlite> select url from urls;
http://download.microsoft.com/download/7/1/7/7179A150-F2D2-4502-9D70-4B59EA148EAA/IE11-Windows6.1-x64-en-us.exe
http://en.wikipedia.org/wiki/Cloud_storage
http://en.wikipedia.org/wiki/Digital_forensics
http://en.wikipedia.org/wiki/Intellectual_property
http://en.wikipedia.org/wiki/List_of_data_recovery_software
http://forensicswiki.org/wiki/Anti-forensic_techniques
http://go.microsoft.com/fwlink/?LinkID=121792
http://go.microsoft.com/fwlink/?LinkId=69157
```

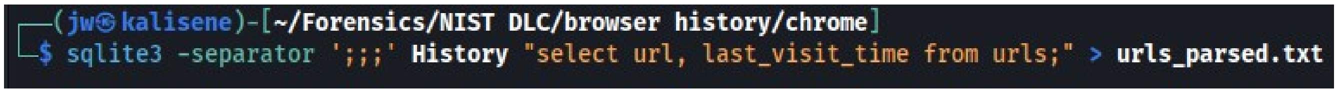
These URLs aren't as easy to read, but they are the literal URLs that were present in the web browser, as opposed to the search terms that the user typed in.



```
sqlite> select last_visit_time from urls;
13071510624000000
13071510676000000
13071510652000000
13071510564000000
```

This timestamp is used by WebKit and Chrome. What makes these timestamps different from one another is their epoch, the date from which they start counting. To convert it to a readable format, **epochconverter.com** has some boilerplate code.

I set the separator to something unique to prevent errors in the final output, like if a specific field has a comma in it.



```
┌──(jw㉿kalisene)-[~/Forensics/NIST DLC/browser history/chrome]
└─$ sqlite3 -separator ';;;' History "select url, last_visit_time from urls;" > urls_parsed.txt
```

Conclusion

We've covered internet history for two versions of Internet Explorer and one for Chrome. IE may not be a common browser these days, but Chrome certainly is. There are existing libraries to make the analysis much easier, but I still had to resort to manual analysis to find evidence of misbehaviour.