Cloud File-Sharing Service Crawler

Cybercrime and Forensics

J.C. Marques R.A.H. Lahaye

June 22, 2018

Abstract

Due to the increasing popularity of Cloud File-Sharing Services, forensic researchers face challenges in the retrieval of forensic artefact as data is more often stored in the cloud than locally. Sufficient evidence of the use of such service is required before a court order can be given to acquire the data stored in the cloud.

The goal of the project is to figure out if the retrieval of evidence is possible, and if so, in a automated manner. This goal resulted in the following research questions: *"Can cloud-services file sync information be used as evidence?"*. This research question has been answered with use of the following sub-questions: *"How can a Cloud file-sharing service be detected?"* and *"Can sufficient artefacts be retrieved to prove that the service has been used by the suspect?"*.

Research shows that known Cloud file-sharing services can be detected by using fuzzy hashing, and that most of these services leave forensic artefacts behind including, but not limited to, email address, sync/root folder, and file records.

As proof of concept, a Cloud File-Sharing Crawler has been built that is able to detect these services, and retrieve its forensic artefacts in an automated manner.

1 Introduction

Cloud services have become more popular than ever in the last few years due to the operability, flexibility, and scalability they provide. This raises new challenges for forensic researches as data can be stored in the cloud instead of locally, and due to the fact that many different cloud services are available, resulting in many different and new forensic artefacts.

Often clients (not necessarily web-clients) are used to access these cloud services leaving artefacts behind. The goal of this project is to detect the use of cloud file-sharing services. Also, to find out if evidence and information can be gathered in an automated way to be of use during a case. Uses such as: aiding the request of court orders to acquire user data directly from the service providers.

This goal resulted in the following research question: *"Can cloud-services file sync information be used as evidence?"*.

This research question has been answered with use of the following sub-questions:

1. How can a cloud file-sharing service be detected?
2. Can sufficient artefacts be retrieved to prove that the service has been used by the suspect?

These research questions are scoped to the following cloud file-sharing clients for Windows:

- Dropbox

- Google Drive
- iCloud Drive
- OneDrive
- Mega Sync

These services were chosen as they are among the most widely used[Drake, 2018].

Forensic artefacts are scoped to:

- Email address or username
- Location of sync/root folders
- File sync information/records

To answer these questions, an in depth literature research on cloud service artefact gathering and decoding was done. When no artefacts were found, a manual analysis was done. Also a program that crawls the filesystem for cloud services and gathers information on found services was built as proof of concept.

2 Literature Review

Fuzzy Hashing Is a type of hashing algorithm that hashes pre-calculated number of blocks of a file individually, and map each hash to a single character of the final hash. This can be used not only to detect exact file matches, but also to make a comparison based on similarity between the two files. This is possible due to the final hash changing only characters the correspond to changed blocks. This enables matching of different versions of the same file independently of changes made to it. This algorithm was discussed at great lengths in the papers of Frank Breitinger and Harald Baier 2012, and Jesse Kornblum 2006. These papers enabled us to have a higher understanding of how fuzzy hashes work, and how they can be used to reduce the amount of hashes we need per cloud service to make an accurate assumption of its existence in a filesystem. Fuzzy hashes where used in our project as a mean to identify cloud services present in the file system.

Optimising ssDeep for use at scale Brian Wallace 2015 wrote a paper where he describes a group of techniques to optimise ssdeep (one of several implementations of fuzzy hashing) to speed up the comparison operations on large data-sets. The core of these ideas revolve around running comparisons only when certain conditions are met, such as comparing only hashes with the same block size. This enables the reduction of unnecessary comparisons. This paper helped us devise ways to also speed up our proof of concept during comparisons of cloud service binaries.

Cloud Storage Forensics Mattia Epifani 2013 has done research on artefacts for cloud services such as; Dropbox, SkyDrive, Google Drive, and iCloud. In his paper M. Epifani describes what artefacts it could find in the folders created during installation as well as environment variables changed and even registry keys added and removed for each service. Although this is a very similar research to what we have attempted, most of the information found was heavily outdated and as such most issues had been corrected/changed for newer versions of the service clients. This research could only be used accurately regarding Google Drive. His analysis of Dropbox is mainly on unencrypted databases found in the outdated versions of the client. SkyDrive has been superseded by OneDrive which came with a large portion of the client rewritten/changed and as such the artefact research was not useful for this research as they did not contain file records or usernames. Regarding iCloud, his analysis was done solely on iCloud backups, and not iCloud Drive.

Brush up on Dropbox DBX decryption Francesco Picasso 2017 wrote a paper where he describes techniques to decrypt Dropbox dbx files both in "live" and "post-mortem" environments. He also created a toolkit to consolidate those techniques in one place and for easier usage. His work has been used to help decrypt the dbx files. This was possible due to Dropbox not having revised or changed its methods of encryption since as far as 2012.

DPAPI offline decryption utility Jordan Tucker created a python toolkit implementation of the Microsoft DPAPI (Data Protection API) called DPAPIck. This enables the cross platform offline decryption of DPAPI structures. This tookit has been used to recover information from a windows filesystem (such as user password), and the decryption of the Dropbox databases.

No further research could be found regarding OneDrive, MegaSync, or iCloud Drive. For these services we extrapolated from the other research the steps we could take to try to accomplish the same result in these services.

3 Forensic Artefacts

Each cloud file-sharing service has its own forensic artefacts. This section contains an analysis of each service's forensic artefacts regarding the used username, file sync/root folder, and file sync information/records.

3.1 Dropbox

Dropbox stores its configuration in the database file "C:/Users/os3/AppData/Local/Dropbox/instance1/config.dbx", and its file cache in "C:/Users/os3/AppData/Local/Dropbox/instance1/filecache.dbx". Both files contain a dbx extension, implying that both are encrypted. Dropbox makes use of the SQLite Encryption Extension (SEE) for SQLite to encrypt the databases. To decrypt, decryption keys are required, keys which are stored in the Windows registry.

These decryption keys are stored as a binary large object (BLOB). A BLOB is a data type that can store binary objects or data. These blobs are encrypted as well by the Data Protection Application Programming Interface of Windows, and are therefore linked to a local Windows user account. To decrypt this blob, the user password or the SHA1 of the user password encoded in UTF16LE is required[Picasso, 2017][Picasso, 2015].

To retrieve the user password, 'lsasecrets' is used from the Windows DPAPI decryption & forensic toolkit, also known as dpapick[Tucker]. lsasecrets require the system and security hive of the Windows registry, stored at "C:/Windows/System32/config/SYSTEM" and "C:/Windows/System32/config/SECURITY"[Picasso, 2015]. Input and output of lsasecrets can be seen in Appendix A.1 - Figure 1.

Decrypting the BLOB results in the so called Dropbox user keys. The user keys cannot be used to decrypt the dbx itself, though the dbx decryption keys are derived from these user keys by using PBKDF2 with 1066 iterations and a fixed salt, 0D638C092E8B82FC452883F95F355B8E. PBKDF2 is a key derivation functions that reduce the vulnerability of encrypted keys to brute force attacks[Wikipedia]. To decrypt the blob automatically, and undo the PBKDF2 key deviation function, 'dbx-key-win-dpapi.py' can be used from the Windows DPAPI laboratory toolkit by dfirfpi[dfirfpi, a]. dbx-key-win-dpapi.py requires the user Security Identifier (SID) and the user password that can be retrieved with lsasecrets. Input and output of dbx-key-win-dpapi.py can be seen in Appendix A.1 - Figure 2.

Using the retrieved dbx encryption keys from dbx-key-win-dpapi.py, the database can be decrypted using 'sqlite-dbx-tux64' from the Dropbox DBX toolkit[dfirfpi, b], or the sqlite shell binary. Input and output of sqlite-dbx-tux64 can be seen in Appendix A.1 - Figure 3.

The decrypted databases are plain sqlite databases and can be viewed with a sqlite client.

The Dropbox email address is stored under the key 'email' in the the table 'config' in the database 'config.dbx', the root folder under the key 'dropbox_path' in the 'config' table as well. An example of these keys can be seen in Appendix A.1 - Figure 4.

Filerecords are stored under the table 'file_journal' in the file 'filecache.dbx'. An example of these file records can be seen in Appendix A.1 - Figure 5.

3.2 Google Drive

Google Drive stores its configuration in the database file "C:/Users/os3/AppData/Local/Google/Drive/user_default/sync_config.db", and its file cache in "C:/Users/os3/AppData/Local/Google/Drive/user_default/snapshot.db". Both files have a .db extension and are plain sqlite databases.

The email address is stored under the key 'user_email' in the table 'data' in the database 'sync_config.db'. The root folder is stored under the key 'root_config__0' in the same table. An example of these keys can be seen in Appendix A.2 - Figure 6.

File records are stored in the database 'snapshot.db'. This database contains 2 tables; local_entry and cloud_entry. The table 'local_entry' contains records of local files, where 'cloud_entry' contains records of remote files on the server. An example of these file records can be seen in Appendix A.2 - Figure 7 and Appendix A.2 - Figure 8.

3.3 iCloud Drive

iCloud Drive stores its configuration in the database file "C:/Users/os3/AppData/Local/AppleInc/CloudKit/iCloudDrive/ckcachedatabase.db", local file cache in "C:/Users/AppData/Local/AppleInc/iCloudDrive/client.db", and remote file cache in "C:/Users/os3/AppData/Local/AppleInc/iCloudDrive/server.db". All three files have a .db extension and are plain sqlite databases.

The email address is stored under the first field of each key in the table 'PCSData' in the database 'ckcachedatabase.db'. An example of this key can be seen in Appendix A.3 - Figure 9

The root folder can be derived by the path of the file records[1]. Local file records are stored in the table 'items' in the database 'client.db', records of the remote files on the server are store in the table 'server_items' in the database 'server.db'. An example of the local file records can be seen in Appendix A.3 - Figure 10, and remote files in Appendix A.3 - Figure 11.

[1] iCloud Drive does not allow changing root folder

3.4 OneDrive

OneDrive stores its configuration under an .ini file with a 16 character long file name that exists out of random numbers and letters in the directory "C:/Users/os3/AppData/Local/Microsoft/OneDrive/settings/Personal". OneDrive also stores the configuration in the registry in the directory "HKEY_CURRENT_USER/Software/Microsoft/OneDrive/Accounts/Personal". This hive is located on the file system as "C:/Users/os3/NTUSER.DAT".

File records are stored in a .dat file in the same directory with the same filename as the configuration file. .dat files are used for application specific encoding, and therefore the coding is unknown.

In the analysis done for the project, the file paths were as following: "C:/Users/os3/AppData/Local/Microsoft/OneDrive/settings/Personal/53fadd265f21ec2a.ini" and "C:/Users/os3/AppData/Local/Microsoft/OneDrive/settings/Personal/53fadd265f21ec2a.dat".

The root folder can be found in the .ini file of Onedrive under the key 'library', or in the user registry under the key 'UserFolder' in the directory "HKEY_CURRENT_USER/Software/Microsoft/OneDrive/Accounts/Personal". The user name is stored in the same directory under the key 'UserEmail'. An example of the configuration in the .ini file can be seen in Appendix A.4 - Figure 12, the registry keys in Appendix A.4 - Figure 13.

File records are stored in the .dat file of Onedrive. Between each file record padding is used as '0xab 0xab 0xab 0xab'.

A file record starts with '0x00 0x00 0x00 name 0x00 0x00 0x00 0xab' where name is the file name in ASCII with each character delimited by '0x00'.

A directory record starts with '0xab 0xab 0xab name 0x00 0x00 0x00 0xab' where name is the directory name in ASCII with each character delimited by '0x00'.

An example of a file and directory record can be seen in Appendix A.4 - Figure 14.

3.5 Mega Sync

Mega Sync saves its configuration under "C:/Users/os3/AppData/Local/MegaLimited/MEGAsync/MEGAsync.cfg". Each value in this configuration file is heavily encrypted[2].

By looking at the source code it seems that each value in configuration file is XORed with the master key. After XORing, the result is XORed again with the hash of the master key.

The master key itself is generated by XORing a fixed seed with a 128bit local generated key. This key is then hashed as SHA1 and represents the master key (keep in mind that the value in the configuration file use the hash of the hash of the master key)[Meganz].

3.6 Comparison

Figure 1 shows a comparison on what artefacts can be found per service.

Service	**Username /E-mail**	**Root Folder**	**File Records**
Dropbox	Yes	Yes	Yes
Google Drive	Yes	Yes	Yes
iCloud Drive	Yes	Yes	Yes
OneDrive	Yes	Yes	Yes
Mega Sync	No	No	No

Table 1: Comparison Forensic Artefacts

[2]No analysis has been done due to limited time and knowledge

4 Proof of Concept

The link to the proof of concept Github page can be found in Appendix B.1.

To create this proof of concept we took the following approach:

1. Create fuzzy hashes for a number of versions of the same cloud service
2. Create modular script:
   - Hash files parsing
   - Database hash merging
   - File system crawling
   - File Hashing and comparing
   - Service databases decryption
   - Service decryption parsing
3. Optimisation of code
4. Testing at each phase and of final product

4.1 Fuzzy Hashing

To create the fuzzy hashes we used ssdeep which takes as input a file path and optionally a block size, and outputs a fuzzy hash with format: "blocksize:hash:hash,filename". To do this we went into each service installation directory and fire the command: *ssdeep -b -r ./* > [filename].* The options:

- -b -> filename field contains only the name without pre-appended file-path
- -r -> hash each file recursively down the directory path
- > [filename] -> the name of the file to output the resulting hashes

The script was created accounting accounting for run-arguments. The first argument has to be the path to a file containing hashes or the path to a directory containing hash files. If the latter argument was used it recursively traverses the directory reading hash files and extracting all of its hashes. These extracted hashes are then merged into a single database file to be used by the script during the same and subsequent runs.

The script also takes by mean of arguments the path to an image mount-point and a list of paths inside this mount-point to search for files for comparison. Once this section of the script starts running it recursively transverses the directories provided, creating and executing threads for each file that it finds, to create hashes for later comparison. This is done with the help of a python library (wrapper for compiled ssdeep binary). Once this is done it compares the hashes created to the hash database. If any match is found it stores what service was found. It then gives the user control of what service gets decrypted. When the user chooses a service the script runs the decryption related to that service and presents at the end the Forensic Artefacts found. An example output can be found in Appendix B.3. Also, for reference the script comes with a fully featured help page that cna be seen with the usage of the –help argument as seen in Appendix B.2

4.2 Forensic Artefacts

For the retrieval of the forensic artefacts of each service, different tools and libraries were used. Most of the artefacts are retrieved through the related sqlite database, therefore the Python library sqlite3 is used which is part of the Python standard library. This library allows the writing and reading of databases[pyt].

4.2.1 Dropbox

Because of the encryption that is used for the databases by Dropbox, multiple tools and scripts were needed. lsasecrets from the Windows DPAPI decryption & forensic toolkit, also known as dpapick[Tucker], was used to retrieve the password of the local user account.

dbx-key-win-dpapi.py from the Windows DPAPI laboratory toolkit by dfirfpi[dfirfpi, a] was used to retrieve the decryption keys for decrypting the Dropbox database. For this process the user password is required.

Next, sqlite-dbx-tux64 from the same toolkit was used to decrypt the encrypted database with the decryption key. After decryption, the following artefacts are retrieved with the sqlite3 library: email address, root folder, and file records.

4.2.2 Google Drive and iCloud Drive

Google Drive and iCloud made use of sqlite databases, and therefore the library sqlite3 was used to retrieve the following artefact: email address, root folder, and file records.

4.2.3 OneDrive

The configuration for OneDrive is saved in the registry. Therefore, the Python library python-registry is used that provides read access to the registry. With use of this library the following artefacts are retrieved: email address and root folder.

Retrieval of file record artefacts have not been implemented in the Cloud File-Sharing Service Crawler.

4.2.4 Mega Sync

Retrieval of Mega Sync forensic artefacts have not been implemented either in the Cloud File-Sharing Service Crawler.

4.2.5 Results

Figure 2 shows a comparison on what artefacts can be retrieved by using the created Cloud File-Sharing Service Crawler.

Service	Username /E-mail	Root Folder	File Records
Dropbox	Yes	Yes	Yes
Google Drive	Yes	Yes	Yes
iCloud Drive	Yes	Yes	Yes
OneDrive	Yes	Yes	No
Mega Sync	No	No	No

Table 2: Retrievable Forensic Artefacts of Cloud File-Sharing Service Crawler

5 Discussion

Can cloud-services file sync information be used as evidence? Yes it can, as we demonstrated with this project. Although the services can be found and user data extracted, it is not without some short comings. Short coming such as; resources usage, accuracy and even time.

Due to relying on walking the directory tree and hashing every single file, a trade-off had to be done where to reduce the time this operation takes we had to increase the amount of memory the script requires to be able to create has many threads as possible to handle the hashing of each file.

Accuracy also is a strong factor since it relies on the hash comparison accurately displaying if the binary/configuration files are present on the system but without the exact matching of normal hashes. Despite the fuzzy hashing allowing for multi-version matching, it brings to the equation a balance between acceptance and rejection which influences the amount of false-positives and false-negatives that are found. For example while fine-tuning this acceptance value it frequently occurred that we could find more matches than we had hashes in the same VM image we extracted the hashes from initially. This means that some other files on the scanned paths were similar enough to trigger as an accepted match. Despite there being a chance for wrong matching/unmatching it allows for a smaller database of cloud services. This means that less work needs to be done and stored for additional services being added to this proof of concept.

We have also arrived to the conclusion that by generating the fuzzy hashes without passing a block-size, the hashes get generated with a block size pre-calculated on the basis of the size of the file. This counteracts one of the optimisation's proposed by Brian Wallace 2015 which is making the comparison only of hashes of the same size. So if a file not only gets partially changed but its size also increases, it would have a different block size. This means comparisons based on same block size would not even be made. Furthermore, even if this optimisation is not used if the block size changes it will change the whole hash since the characters will be mapped to a different number and place of bytes in the stream.

The decryption process of some of the cloud services that rely on the DPAPI descryption is only possible from Windows XP until Windows 8.1. Which means criminal cases involving Windows 10 and Windows servers might not yield any data for these services.

Full disk encryption was found to be a problem since an extracted image would be protected from praying eyes unless the password is known. This can only be mitigated in an analyst would know the password in which case would be take care at mount time where the partition would be decrypted and mounted.

We could not implement the decryption of Mega Sync due to the short time frame of the project in conjunction with the increased complexity of the task. For this reason we prioritised the other services that were more accessible. Also, in OneDrive despite being able to get to the files with the information, these files are encoded with application specific encoding which means there is no accurate way

to decode them. We did manage to get some information and indicators of the information present but not enough to accurately state that it was decoded.

There is also an ongoing issue with opening sqlite databases in read-only mount-points where the sqlite3 python library needs write access to the database file to be able to open it. For this reason only when mounted in read/write the database can be read and queried. This problem seems to be only present on python2.7 code and as such should work out of the box on python3.

Lastly, it was not possible to implement in the short time frame of this project a way to find platform independent paths to database files. For this reason the paths for the decryption were hard-coded for a Windows system with default paths for the installation. Ideally the paths would be automatically found and used on the decryption process since a user can change its installation directory.

Despite these short-comings this project shows the possibilities and opens up a path for improvement. With some work and out of the box thinking it should be possible to automate the whole process enabling an analyst to fire the tool on a case image and accurately retrieve user data to be used in the investigation case.

6 Conclusion

Fuzzy Hashing can be used to detect known Cloud file-sharing services by scanning a mount path for known fuzzy hashes. It requires that the fuzzy hashes are known, and therefore it is first needed to map the service's files. The more versions are mapped, the larger the database, the more precise matches can be found.

Dropbox, Google Drive, iCloud Drive, and OneDrive leave forensics artefacts behind including, but not limited to, email address or username, location of the sync/root folder, and file sync information. By crawling these artefacts it can be concluded that a particular user, or potential suspect, has made use of the related service. This evidence can be provided to a judge to receive a court order to acquire the suspect's cloud data. Due to the heavily use of encrypted no forensic artefacts for Mega Sync could be retrieved.

Using the created Cloud Service Crawler one can scan a mount point to detect the known Cloud file-sharing services, and retrieve the needed forensic artefacts to gather proof for the the use of the service. Artefacts that can be retrieved with the crawler are username or email address, sync/root folder, and file records.

7 Future Work

For future work one could extend the crawler to retrieve more forensic artefacts. As this project was solely based on retrieving sufficient artefacts to simply prove if a user has used the service or not, retrieved artefacts are limited.

One could also look into the decryption of the Mega Sync encryption. Due to limited time and knowledge we were unable to create a crawler that is able to retrieve Mega Sync artefacts.

Another recommendation for future work is rewriting the output of the crawler and the file record information so that data is gathered of which a time-line can be created. A time-line is deemed more reliable as proof that simply records with a time-stamp.

As last, one could port the crawler to Python 3 as now only Python 2 is supported. Python 2 has received its end of life date and will not be maintained after the year 2020.

References

sqlite3 db-api 2.0 interface for sqlite databases. URL `https://docs.python.org/2/library/sqlite3.html#module-sqlite3`.

Frank BreitinJordan Tuckerger and Harald Baier. A fuzzy hashing approach based on random sequences and hamming distance. In *Proceedings of the conference on digital forensics, security and law*, page 89. Association of Digital Forensics, Security and Law, 2012.

dfirfpi. Windows dpapi laboratory, a. URL `https://github.com/dfirfpi/dpapilab`.

dfirfpi. A sort of a toolkit to decrypt dropbox windows dbx files, b. URL `https://github.com/dfirfpi/decwindbx`.

Nate Drake. The best cloud storage, 2018. URL `https://www.techradar.com/news/the-best-cloud-storage`.

Mattia Epifani. Cloud storage forensics, 2013. URL `https://www.sans.org/summit-archives/file/summit-archive-1493920922.pdf`.

Jesse Kornblum. Fuzzy hashing. *August*, 21:2006–08, 2006.

Meganz. Easy automated syncing between your computers and your mega cloud drive. URL `https://github.com/meganz/MEGAsync`.

Francesco Picasso. Happy dpapi!, 2015. URL `http://blog.digital-forensics.it/2015/01/happy-dpapi.html`.

Francesco Picasso. Brush up on dropbox dbx decryption, 2017. URL `https://http://blog.digital-forensics.it/2017/04/brush-up-on-dropbox-dbx-decryption.html`.

Jordan Tucker. Dpapi offline decryption utility. URL `https://github.com/jordanbtucker/dpapick`.

Brian Wallace. Optimizing ssdeep for use at scale, 2015. URL `https://www.virusbulletin.com/virusbulletin/2015/11/optimizing-ssdeep-use-scale/`.

Wikipedia. Pbkdf2. URL `https://en.wikipedia.org/wiki/PBKDF2`.

A Forensic Artifacts

A.1 Dropbox

```
root@Sioux-Falls:~# ./lsasecrets --system=/mnt/Windows/System32/config/SYSTEM
    --security=/mnt/Windows/System32/config/SECURITY --secret=DefaultPassword
Welkom123
```

Figure 1: lsasecrets input and output

```
root@Sioux-Falls:~# ./dbx-key-win-dpapi.py --masterkey=/mnt/Users/os3/AppData/
    Roaming/Microsoft/Protect/S-1-5-21-917910545-2581931138-2352614243-1000 --
    sid=S-1-5-21-917910545-2581931138-2352614243-1000 --password=Welkom123 --
    ntuser=/mnt/Users/os3/NTUSER.DAT
------------------------------------------------------------------------------
[ks] user key:  27eb6a64bad26d7b14db8c6707ae7fe1
[ks]  DBX key:  ae16e208df870ca350d81d5993aa8fdb
------------------------------------------------------------------------------
[ks1] user key: 27eb6a64bad26d7b14db8c6707ae7fe1
[ks1]  DBX key: ae16e208df870ca350d81d5993aa8fdb
------------------------------------------------------------------------------
```

Figure 2: dbx-key-win-dpapi.py input and output

```
root@Sioux-Falls:~# ./Dropbox/sqlite-dbx-tux64 -key
    ae16e208df870ca350d81d5993aa8fdb /mnt/Users/os3/AppData/Local/Dropbox/
    instance1/config.dbx ".backup /tmp/config.db"
```

Figure 3: sqlite-dbx-tux64 input and output

```
root@Sioux-Falls:~# ./sqlite3 /tmp/config.db -cmd "SELECT * FROM config;"
...
email|rick.lahaye@os3.nl
userdisplayname|os3 os3
dropbox_path|C:/Users/os3/Dropbox
...
root@Sioux-Falls:~# ./sqlite3 /tmp/config.db -cmd ".schema"
CREATE TABLE config (key TEXT PRIMARY KEY NOT NULL, value BLOB);
```

Figure 4: Dropbox config.db

```
root@Sioux-Falls:~# ./sqlite3 /tmp/filecache.db -cmd "SELECT * FROM
    file_journal;"
1|2604326896:/getting started with dropbox paper.url|2604326896:/|||4|1|
    Getting Started with Dropbox Paper.url|
    kPIn4Te4OtTntzDqxmIUxueJOLhkWYXZu2dHeK50XJc
    ||240|1520512908|1520512908|0|{"dropbox_mute": {"mute_key": {"data": "
    AQAAAAAAAAAA"}, "mute": {"data": "MQ
    =="}}}|1520512908|81554767|1|||||||||||
2|2604326896:/getting started with dropbox.pdf|2604326896:/|||2|1|Getting
    Started with Dropbox.pdf|LiKDtWYdKc4fsgRAZdok9KHJRrzfjj8Dkx4Kh2Q0Ooo
    ||1467194|1520512907|1520512907|0|{"dropbox_mute": {"mute_key": {"data": "
    AQAAAAAAAAAA"}, "mute": {"data": "MQ
    =="}}}|1520512908|81554767|1|||||||||||
root@Sioux-Falls:~# ./sqlite3 /tmp/filecache.db -cmd ".schema"
CREATE TABLE file_journal (id INTEGER PRIMARY KEY NOT NULL, server_path TEXT
    NOT NULL UNIQUE, parent_path TEXT NOT NULL, extra_pending_details
    PENDINGDETAILS2, force_reconstruct INTEGER CHECK (force_reconstruct=1),
    local_sjid INTEGER, local_host_id INTEGER, local_filename TEXT,
    local_blocklist BYTETEXT CHECK(local_blocklist IS NULL OR TYPEOF(
    local_blocklist) == 'text'), local_infinite_details INFINITEDETAILS,
    local_size INTEGER, local_mtime INTEGER, local_ctime INTEGER, local_dir
    INTEGER, local_attrs ATTRIBUTETEXT, local_timestamp INTEGER, local_user_id
    INTEGER, local_sync_type INTEGER, updated_sjid INTEGER, updated_host_id
    INTEGER, updated_filename TEXT, updated_blocklist BYTETEXT CHECK(
    updated_blocklist IS NULL OR TYPEOF(updated_blocklist) == 'text'),
    updated_size INTEGER, updated_mtime INTEGER, updated_dir INTEGER,
    updated_timestamp INTEGER, updated_user_id INTEGER, updated_attrs
    ATTRIBUTETEXT, updated_sync_type INTEGER);
```

Figure 5: Dropbox filecache.db

A.2 Google Drive

```
root@Sioux-Falls:~# ./sqlite3 /mnt/Users/os3/AppData/Local/Google/Drive/
    user_default/sync_config.db -cmd "SELECT * FROM data;"
...
user_email|value|rick.joao.os3@gmail.com
root_config__0|rowkey|C:/Users/os3/Google Drive
..
root@Sioux-Falls:~# ./sqlite3 /mnt/Users/os3/AppData/Local/Google/Drive/
    user_default/sync_config.db -cmd ".schema"
CREATE TABLE data (entry_key TEXT, data_key TEXT, data_value TEXT, UNIQUE (
    entry_key, data_key, data_value));
```

Figure 6: Google Drive sync_config.db

```
root@Sioux-Falls:~# ./sqlite3 /mnt/Users/os3/AppData/Local/Google/Drive/
    user_default/snapshot.db -cmd "SELECT * FROM local_entry"
...
22236523160209878|serial:1811966073|test.txt|1521406662|
    c8dc8fdf774bb5468b9e9cc09eb48384|4|0
7881299347968609|serial:1811966073|test.txt|1522077162|
    a8f5f167f44f4964e6c998dee827110c|6|0
...
root@Sioux-Falls:~# ./sqlite3 /mnt/Users/os3/AppData/Local/Google/Drive/
    user_default/snapshot.db -cmd ".schema"
CREATE TABLE local_entry (inode INTEGER NOT NULL, volume TEXT NOT NULL,
    filename TEXT, modified INTEGER, checksum TEXT, size INTEGER, is_folder
    INTEGER, PRIMARY KEY (inode, volume));
```

Figure 7: Google Drive snapshot.db Local Entries

```
root@Sioux-Falls:~# ./sqlite3 /mnt/Users/os3/AppData/Local/Google/Drive/
    user_default/snapshot.db -cmd "SELECT * FROM cloud_entry"
...
1VsTpfv0fe_hbYQ9OgeLUHw-etQlyiCsB|test.txt|1521406662||0|1|0|4|
    c8dc8fdf774bb5468b9e9cc09eb48384|0||||
1H8HMv9xQgy7OGCRXH6fR4cS4S5mDT4Pb|test.txt|1522077162||0|1|0|6|
    a8f5f167f44f4964e6c998dee827110c|0||||0
...
root@Sioux-Falls:~# ./sqlite3 /mnt/Users/os3/AppData/Local/Google/Drive/
    user_default/snapshot.db -cmd ".schema"
CREATE TABLE cloud_entry (doc_id TEXT NOT NULL, filename TEXT, modified
    INTEGER, created INTEGER, acl_role INTEGER, doc_type INTEGER, removed
    INTEGER, size INTEGER, checksum TEXT, shared INTEGER, resource_type TEXT,
    original_size INTEGER, original_checksum TEXT, down_sample_status INTEGER,
     PRIMARY KEY (doc_id));
```

Figure 8: Google Drive snapshot.db Cloud Entries

A.3 iCloud Drive

```
root@Sioux-Falls:~# ./sqlite3 /mnt/Users/os3/AppData/Local/"Apple Inc"/
    CloudKit/iCloudDrive/ckcachedatabase.db -cmd "SELECT * FROM PCSData"
...
rick.lahaye@os3.nl|com.apple.clouddocs|57T9237FN3.net.whatsapp.WhatsApp:
    __defaultOwner__|2|1|
...
root@Sioux-Falls:~# ./sqlite3 /mnt/Users/os3/AppData/Local/"Apple Inc"/
    CloudKit/iCloudDrive/ckcachedatabase.db -cmd ".schema"
CREATE TABLE PCSData (accountID TEXT, containerName TEXT, identifier TEXT,
    databaseScope INTEGER, itemType INTEGER, CKDPCSData BLOB, zoneID TEXT,
    shareID TEXT, parentID TEXT, date INTEGER, PRIMARY KEY(accountID,
    containerName, identifier));
```

Figure 9: iCloud Drive ckcachedatabase.db

```
root@Sioux-Falls:~# ./sqlite3 /mnt/Users/os3/AppData/Local/"Apple Inc"/
    iCloudDrive/client.db -cmd "SELECT * FROM items;"
2|3430C022-67BA-4FFF-8E67-FBF7751BE003|__defaultOwner__
    |0|0||0|0|0|0||0|0|0|0|0|28147497671857 95|0|test2.txt.txt||C:\Users\os3\
    iCloudDrive\test2.txt.txt|0|0||42|0|1|1|0|1522083655|root|test2.txt.txt
    |||||||41|1522083658|test2.txt.txt|5|0||3 'G
g     %ys_%      z z ||||||0|0|0
2|C4B39FD3-84EA-4A01-8365-A1049CC7EC9C|__defaultOwner__
    |0|0||0|0|0|0||0|0|0|0|0|53480245575817 76|0|test.txt||C:\Users\os3\
    iCloudDrive\test.txt|0|0||45|0|1|1|0|1522083655|root|test.txt
    |||||||44|1522083665|test.txt|4|0||    '        3  '    N    y J
    ||||||0|0|0
root@Sioux-Falls:~# ./sqlite3 /mnt/Users/os3/AppData/Local/"Apple Inc"/
    iCloudDrive/client.db -cmd ".schema"
CREATE TABLE IF NOT EXISTS 'items' ( zone_row_id integer not null, item_id
    blob , owner_id text not null, sharing_options integer, record_permission
    integer, share_id text not null, item_localsyncupstate integer,
    item_in_flight_diffs integer, item_local_diffs integer not null,
    item_notifs_rank integer not null, item_desired_version_etag text default
    null, item_desired_version_size integer, item_desired_version_rank integer
    , item_needs_app_upgrade integer, item_processing_stamp integer,
    item_doc_id integer, item_file_id integer, item_generation integer,
    item_localname text, item_bouncedname text, item_absolutepath text,
    item_staged_file_id integer, item_staged_generation integer,
    item_staged_manifest_file_id integer, item_stat_etag blob default null,
    item_state integer not null, item_type integer not null, item_mode integer
     not null, item_creator_id integer not null, item_birthtime integer not
    null, item_parent_id blob not null, item_filename text not null,
    item_finder_info blob, item_additions_ck blob, item_public_sharing_key
    blob, item_public_sharing_base_token text, item_target_id text,
    item_override_filename text, version_etag blob default null, version_mtime
     integer default 0, version_name text, version_size integer default 0,
    version_thumbnail_size integer default 0, version_thumbnail_signature blob
     default null, version_content_signature blob default null,
    version_manifest_signature blob default null, version_additions_ck blob
    default null, version_public_sharing_key blob, version_uploaded_assets
    blob default null, desired_version blob default null, upload_error_code
    integer not null default 0, upload_retry_count integer not null default 0,
     upload_retry_time integer not null default 0, PRIMARY KEY (zone_row_id,
    item_id ASC));
```

Figure 10: iCloud Drive client.db

```
root@Sioux-Falls:~# ./sqlite3 /mnt/Users/os3/AppData/Local/"Apple Inc"/
    iCloudDrive/server.db -cmd "SELECT * FROM server_items;"
2|3430C022-67BA-4FFF-8E67-FBF7751BE003|__defaultOwner__
    |0|1||2|42|0|1|1|0|1522083655|root|test2.txt.txt|||42|||||41|1522083658|
    test2.txt.txt|5|0||3 'G
g     %ys_%      z z |||41||2|-1
2|C4B39FD3-84EA-4A01-8365-A1049CC7EC9C|__defaultOwner__
    |0|1||3|45|0|1|1|0|1522083655|root|test.txt|||45|||||44|1522083665|test.
    txt|4|0||     '        3 '    N    y J           |||44||2|-1
root@Sioux-Falls:~# ./sqlite3 /mnt/Users/os3/AppData/Local/"Apple Inc"/
    iCloudDrive/server.db -cmd ".schema"
CREATE TABLE IF NOT EXISTS 'server_items' ( zone_row_id integer not null,
    item_id blob not null, owner_id text not null, sharing_options integer,
    record_permission integer, share_id text not null, item_rank integer
    default 0, item_stat_etag blob, item_state integer not null, item_type
    integer not null, item_mode integer not null, item_creator_id integer not
    null, item_birthtime integer not null, item_parent_id blob not null,
    item_filename text not null, item_finder_info blob, item_additions blob,
    item_additions_ck blob, item_public_sharing_key blob,
    item_public_sharing_base_token text, item_target_id text,
    item_override_filename text, version_etag blob, version_mtime integer,
    version_name text, version_size integer, version_thumbnail_size integer,
    version_thumbnail_signature blob, version_content_signature blob,
    version_manifest_signature blob, version_additions blob,
    version_additions_ck blob, version_public_sharing_key blob, item_scope,
    integer default -1, PRIMARY KEY (zone_row_id, item_id ASC), UNIQUE (
    item_rank));
```

Figure 11: iCloud Drive server.db

A.4 iCloud Drive

```
root@Sioux-Falls:~# head -1 /mnt/Users/os3/AppData/Local/Microsoft/OneDrive/
    settings/Personal/53fadd265f21ec2a.ini
library = 1 4 53FADD265F21EC2A!101 1520518448 "SkyDrive" Me personal "C:/Users
    /os3/OneDrive" 1
```

Figure 12: OneDrive 53fadd265f21ec2a.ini

```
root@Sioux-Falls:~# chntpw -e /mnt/Users/os3/NTUSER.DAT
cat /Software/Microsoft/OneDrive/Accounts/Personal/UserFolder
Value </Software/Microsoft/OneDrive/Accounts/Personal/UserFolder> of type
    REG_SZ (1), data length 44 [0x2c]
C:/Users/os3/OneDrive
cat /Software/Microsoft/OneDrive/Accounts/Personal/UserEmail
Value </Software/Microsoft/OneDrive/Accounts/Personal/UserEmail> of type
    REG_SZ (1), data length 52 [0x34]
rick.joao.os3@outlook.com
```

Figure 13: OneDrive Registry

```
root@Sioux-Falls:~# hexdump -C /mnt/Users/os3/AppData/Local/Microsoft/OneDrive
    /settings/Personal/53fadd265f21ec2a.dat
...
00000b50  04 00 00 00 00 00 00 00  74 00 65 00 73 00 74 00  |........t.e.s.t.|
00000b60  2e 00 74 00 78 00 74 00  00 00 ab ab ab ab ab ab  |..t.x.t.........|
00000b70  ab ab ab ab ab ab ab ab  ab ab ab ab ab ab ab ab  |................|
...
00000f60  8e 09 01 00 00 00 2b 00  00 ab 00 00 ab ab ab ab  |......+.........|
00000f70  74 00 65 00 73 00 74 00  5f 00 66 00 6f 00 6c 00  |t.e.s.t._.f.o.l.|
00000f80  64 00 65 00 72 00 00 00  ab ab ab ab ab ab ab ab  |d.e.r...........|
00000f90  ab ab ab ab ab ab ab ab  ab ab ab ab ab ab ab ab  |................|
...
000013c0  05 00 00 00 00 00 00 00  74 00 65 00 73 00 74 00  |........t.e.s.t.|
000013d0  32 00 2e 00 74 00 78 00  74 00 00 00 ab ab ab ab  |2...t.x.t.......|
000013e0  ab ab ab ab ab ab ab ab  ab ab ab ab ab ab ab ab  |................|
```

Figure 14: OneDrive File and Directory Record

B Proof of Concept

B.1 Github Page

`https://github.com/ricklahaye/Cloud-Service-Crawler`

B.2 PoC Help Page

```
usage: python crawler.py (-f FILE | -d DIR) -t TARGET

This script is a cloud service crawler that finds commonly used cloud services
in a path.
```

```
optional arguments:
  -h, --help            show this help message and exit
  -u USER, --user USER  Specify user
  -m MOUNT_POINT, --mount-point MOUNT_POINT
                        Specify Mountpoint of the filesystem
  -s SCAN_DIRECTORY, --scan-directory SCAN_DIRECTORY
                        Specify directories to scan
  -v, --verbose         Show debugging information
  --version             show program's version number and exit

Required:
  -f FILE, --file FILE  Path to hashes file
  -d DIR, --dir DIR     Path to directory containing hashes files
```

B.3 Output of the PoC

```
FuzzyHashes in database:
  Dropbox:827
  Onedrive:1324
  Google:142
  Megasync:80
  Icloud:962

FuzzyHashes matches:
  Dropbox:325
  Onedrive:15
  Google:71
  Megasync:0
  Icloud:405

2018-04-09 13:02:09,055 - root.decryptor - INFO - workers got access to settings and initalized the logg
Options Available:
1. dropbox
2. google
3. onedrive
4. icloud
5. Quit
Choose one of the options (by number):1

==================== Dropbox ====================
Dropbox Local User Account Password: Welkom123
Dropbox Local User Account SID: S-1-5-21-917910545-2581931138-2352614243-1000
--------------------------------------------------------------------------------
[ks] user key: 27eb6a64bad26d7b14db8c6707ae7fe1
[ks]  DBX key: ae16e208df870ca350d81d5993aa8fdb
--------------------------------------------------------------------------------
```

```
[ks1] user key: 27eb6a64bad26d7b14db8c6707ae7fe1
[ks1]  DBX key: ae16e208df870ca350d81d5993aa8fdb
--------------------------------------------------------------------------------

[u'email', u'rick.lahaye@os3.nl']
[u'userdisplayname', u'os3 os3']
[u'dropbox_path', u'C:\\Users\\os3\\Dropbox']

* Files according to Dropbox file cache:
[u'2604326896:/getting started with dropbox paper.url', u'Getting Started with Dropbox Paper.url']
[u'2604326896:/getting started with dropbox.pdf', u'Getting Started with Dropbox.pdf']
==================== Dropbox ====================
Options Available:
1. dropbox
2. google
3. onedrive
4. icloud
5. Quit
Choose one of the options (by number):2
==================== Google ====================
/mnt/Users/os3/AppData/Local/Google/Drive/user_default/sync_config.db
[u'root_config__0', u'C:\\Users\\os3\\Google Drive']
[u'user_email', u'rick.joao.os3@gmail.com']

/mnt/Users/os3/AppData/Local/Google/Drive/user_default/snapshot.db
* Files on disk according to Google Drive:
[u'\\\\?\\C:\\Users\\os3\\Google Drive', 'Modified: unknown', 'Folder']
[u'\\\\?\\C:\\Users\\os3\\Pictures', 'Modified: unknown', 'Folder']
[u'\\\\?\\C:\\Users\\os3\\Desktop', 'Modified: unknown', 'Folder']
[u'\\\\?\\C:\\Users\\os3\\Documents', 'Modified: unknown', 'Folder']
[u'MEGA', 'Modified: unknown', 'Folder']
[u'Welcome to MEGA.pdf', 'Modified: 2018-02-22 23:01:08', 'File']
[u'Rubbish', 'Modified: unknown', 'Folder']
[u'tmp', 'Modified: unknown', 'Folder']
[u'lock', 'Modified: 2018-03-08 15:19:14', 'File']
[u'OneDrive_v17.3.6943.0625.exe', 'Modified: 2018-03-12 15:48:23', 'File']
[u'OneDrive_v17.3.6998.0830.exe', 'Modified: 2018-03-12 15:47:14', 'File']
[u'Google_Drive_v3.35.5978.2967.msi', 'Modified: 2018-03-18 19:08:49', 'File']
[u'Google_Drive_v3.35.6251.4621.msi', 'Modified: 2018-03-18 19:08:35', 'File']
[u'OneDrive_v17.3.6799.0327.exe', 'Modified: 2018-03-12 15:48:39', 'File']
[u'Dropbox_v42.4.114.exe', 'Modified: 2018-03-12 15:04:37', 'File']
[u'Dropbox_v43.4.49.exe', 'Modified: 2018-03-12 15:04:22', 'File']
[u'iCloudSetup4.0.3.56.exe', 'Modified: 2018-03-18 19:15:26', 'File']
[u'Dropbox_v43.4.50.exe', 'Modified: 2018-03-12 14:15:20', 'File']
[u'iCloudSetup.exe', 'Modified: 2018-03-18 19:19:14', 'File']
[u'OneDrive_v17.3.6517.0809.exe', 'Modified: 2018-03-12 15:48:47', 'File']
```

```
[u'Google_Drive_v3.40.exe', 'Modified: 2018-03-18 19:06:27', 'File']
[u'Dropbox.lnk', 'Modified: 2018-03-18 20:27:29', 'File']
[u'Google_Drive_v3.39.exe', 'Modified: 2018-03-18 19:04:58', 'File']
[u'OneDrive_v17.3.6917.0607.exe', 'Modified: 2018-03-12 15:48:32', 'File']
[u'Google_Drive_v3.36.6721.3394.msi', 'Modified: 2018-03-18 19:08:18', 'File']
[u'Dropbox_v41.4.80.exe', 'Modified: 2018-03-12 15:05:01', 'File']
[u'test.txt', 'Modified: 2018-03-18 21:57:42', 'File']
[u'New_folder', 'Modified: unknown', 'Folder']
[u'test.txt', 'Modified: 2018-03-18 21:57:42', 'File']

* Cloud files last seen on server according to Google Drive:
[u'root', 'Modified: unknown', 'Folder']
[u'Pictures', 'Modified: unknown', 'Folder']
[u'Desktop', 'Modified: unknown', 'Folder']
[u'Documents', 'Modified: unknown', 'Folder']
[u'MEGA', 'Modified: 2018-03-18 20:25:45', 'Folder', 'Removed: no']
[u'Welcome to MEGA.pdf', 'Modified: 2018-02-22 23:01:08', 'File', 'Removed: no']
[u'Rubbish', 'Modified: 2018-03-08 15:19:14', 'Folder', 'Removed: no']
[u'tmp', 'Modified: 2018-03-08 15:19:14', 'Folder', 'Removed: no']
[u'lock', 'Modified: 2018-03-08 15:19:14', 'File', 'Removed: no']
[u'OneDrive_v17.3.6943.0625.exe', 'Modified: 2018-03-12 15:48:23', 'File', 'Removed: no']
[u'OneDrive_v17.3.6998.0830.exe', 'Modified: 2018-03-12 15:47:14', 'File', 'Removed: no']
[u'Google_Drive_v3.35.5978.2967.msi', 'Modified: 2018-03-18 19:08:49', 'File', 'Removed: no']
[u'Google_Drive_v3.35.6251.4621.msi', 'Modified: 2018-03-18 19:08:35', 'File', 'Removed: no']
[u'OneDrive_v17.3.6799.0327.exe', 'Modified: 2018-03-12 15:48:39', 'File', 'Removed: no']
[u'Dropbox_v42.4.114.exe', 'Modified: 2018-03-12 15:04:37', 'File', 'Removed: no']
[u'Dropbox_v43.4.49.exe', 'Modified: 2018-03-12 15:04:22', 'File', 'Removed: no']
[u'iCloudSetup4.0.3.56.exe', 'Modified: 2018-03-18 19:15:26', 'File', 'Removed: no']
[u'Dropbox_v43.4.50.exe', 'Modified: 2018-03-12 14:15:20', 'File', 'Removed: no']
[u'iCloudSetup.exe', 'Modified: 2018-03-18 19:19:14', 'File', 'Removed: no']
[u'OneDrive_v17.3.6517.0809.exe', 'Modified: 2018-03-12 15:48:47', 'File', 'Removed: no']
[u'Google_Drive_v3.40.exe', 'Modified: 2018-03-18 19:06:27', 'File', 'Removed: no']
[u'Dropbox.lnk', 'Modified: 2018-03-18 20:27:29', 'File', 'Removed: no']
[u'Google_Drive_v3.39.exe', 'Modified: 2018-03-18 19:04:58', 'File', 'Removed: no']
[u'OneDrive_v17.3.6917.0607.exe', 'Modified: 2018-03-12 15:48:32', 'File', 'Removed: no']
[u'Google_Drive_v3.36.6721.3394.msi', 'Modified: 2018-03-18 19:08:18', 'File', 'Removed: no']
[u'Dropbox_v41.4.80.exe', 'Modified: 2018-03-12 15:05:01', 'File', 'Removed: no']
[u'test.txt', 'Modified: 2018-03-18 21:57:42', 'File', 'Removed: no']
[u'New_folder', 'Modified: 2018-03-18 21:58:06', 'Folder', 'Removed: no']
[u'test.txt', 'Modified: 2018-03-18 21:57:42', 'File', 'Removed: no']
==================== Google ====================
Options Available:
1. dropbox
2. google
3. onedrive
4. icloud
```

```
5. Quit
Choose one of the options (by number):3

==================== OneDrive ====================
OneDrive user folder: C:/Users/os3/OneDrive
OneDrive user email: rick.joao.os3@outlook.com
==================== OneDrive ====================
Options Available:
1. dropbox
2. google
3. onedrive
4. icloud
5. Quit
Choose one of the options (by number):4
==================== iCloud ====================
User: rick.lahaye@os3.nl

* Files according to Local file cache:
[u'QR website.eps', u'C:\\Users\\os3\\iCloudDrive\\QR website.eps']
[u'test2.txt.txt', u'C:\\Users\\os3\\iCloudDrive\\test2.txt.txt']
[u'test.txt', u'C:\\Users\\os3\\iCloudDrive\\test.txt']

* Files according to Server file cache:
[u'QR website.eps']
[u'test2.txt.txt']
[u'test.txt']
==================== iCloud ====================
Options Available:
1. dropbox
2. google
3. onedrive
4. icloud
5. Quit
Choose one of the options (by number):5
```