FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Department Informatik

Technical Reports / ISSN 2191-5008

Regine Büter, German Engst, Kai Esser, Felix Freiling, Kay Friedrich, Tim Friedrich, Andreas Hammer, Lukas Heine, Lukas Heinlein, Oliver Korn, Christian Minuth, Manuel Müller, Nico Müller, Juliane Reithmeier, Nora Ripley, Matti Schulze, Merlin Sievers, Charinee Srikhaolan, Johan Zenk

A Comparison of Cloud Storage Technologies as Sources of Digital Evidence

Technical Report CS-2022-01

December 2022

Please cite as:
Regine Büter, German Engst, Kai Esser, Felix Freiling, Kay Friedrich, Tim Friedrich, Andreas Hammer, Lukas Heine, Lukas Heinlein, Oliver Korn, Christian Minuth, Manuel Müller, Nico Müller, Juliane Reithmeier, Nora Ripley, Matti Schulze, Merlin Sievers, Charinee Srikhaolan, Johan Zenk , ,,A Comparison of Cloud Storage Technologies as Sources of Digital Evidence," Friedrich-Alexander-Universität Erlangen-Nürnberg, Dept. of Computer Science, Technical Reports, CS-2022-01, December 2022.

Friedrich-Alexander-Universität Erlangen-Nürnberg
Department Informatik

Martensstr. 3 · 91058 Erlangen · Germany

www.cs.fau.de

Contents

A Comparison of Cloud Storage Technologies as Sources of Digital Evidence

Regine Büter, German Engst, Kai Esser, Felix Freiling, Kay Friedrich, Tim Friedrich, Andreas Hammer, Lukas Heine, Lukas Heinlein, Oliver Korn, Christian Minuth, Manuel Müller, Nico Müller, Juliane Reithmeier, Nora Ripley, Matti Schulze, Merlin Sievers, Charinee Srikhaolan, Johan Zenk

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

Abstract

Due to their ease of use and their reliability, managed storage services "in the cloud" have become a standard way to store personal files for many users. In fact, many apps on mobile devices use local storage on the client merely as a cache for data that is fully stored on a remote server. Consequently, data from cloud storage services is an increasingly valuable source of digital evidence in forensic investigations. This document presents the results of a student project that was performed at Friedrich-Alexander-Universität Erlangen-Nürnberg in the winter term 2021/22. Six groups of students analyzed the most relevant network storage technologies (Samba, Nextcloud, Dropbox, Google Drive, OneDrive, iCloud) regarding two questions: (1) What effect does data acquisition by the client have on the data stored on the server? (2) Does the technology support delayed verification of data acquisition in any way? The two questions refer to critical aspects of forensic evidence collection, namely in what way does evidence collection interfere with the evidence, and how easy is it to prove the provenance of data in a forensic investigation. In the concluding discussion we compare the different technologies and develop a taxonomy of storage services that can be used to assess other cloud storage services with regarding the evidental value of data acquired from them.

1. Introduction

Digital evidence, i.e. data that is relevant in a criminal investigation [5], is a common form of evidence today. For many years, the common source of digital evidence was persistent storage, i.e. hard disk drives and later also thumb drives based on flash storage. The domain of digital forensic science [24] was established to study methods and tools to handle digital evidence. For example, borrowing from rules for physical evidence [28, 15], the acquisition of digital evidence was supposed to happen in a manner that preserved as much of the state that the data had when it was collected. Similarly, the collection of digital evidence required proper documentation of where the data came from, when it was acquired and how the acquisition was done (data provenance [22]). This was done to protect the data's *evidential value* [13]. The acquisition of data from persistent storage devices is a well understood area [4], and the applications of the above rules belong to the established set of best-practices in digital forensics.

The increase in (especially mobile) network bandwidth combined with the success of smartphones and other mobile communication devices has catalyzed an economic phenomenon that scholars nowadays refer to as platform economy. Briefly spoken, a platform is an environment in which two market participants (buyers and sellers) come together to commercially interact [23]. Today, such digital ecosystems often built around an operating system (like Microsoft Windows or Apple iOS) or online networked environments built around a marketplace (Amazon), a search engine (Google) or an online social network (Facebook). A central aspect of these platforms is that its data is commonly managed and stored not on the client

device but on network servers. In fact, many apps on mobile devices use local storage on the client merely as a cache for data that is fully stored on a remote server, a concept that has been generalized with the term *cloud computing*. Consequently, data from network storage services is an increasingly valuable source of digital evidence in forensic investigations. In fact, it is not hard to predict that remote data will soon become the dominating part of digital evidence in the courtroom.

While evidence that was acquired from physical storage media could still be regarded as a small adaption of the acquisition procedures for (non-digital) physical evidence, the acquisition of digital evidence from networked storage does not fit easily into classical paradigms of evidence collection. To complicate things further, evidence acquisition over networks creates a multitude of legal problems since data very often does not reside in the same legislative environment where it is needed, and trans-national investigative activities are known to be cumbersome.

1.1. Taxonomy of forensic access to network storage

Remote storage of data over networks is a well-established field today and comes in many different forms. Early approaches such as Sun Microsystems' *Network File System* (NFS) [14] or Microsoft's *Server Message Block* (SMB) [7] were an attempt to simulate accesses to a local file system over the network without having to transfer entire files (as was the idea behind protocols like FTP). Since about 10 years, and arguably popularized through a company called Dropbox Inc., many proprietary network storage services evolved that could be operated through the browser. Modern storage services are often integrated into the systems software of the client and allow seamless synchronization of files over multiple clients.

Access to files on network storage is usually guarded by some form of authentication. Acquiring evidence from network storage is therefore easy if proper authentication credentials like username and password are available to investigators. We call this the *cooperative* model of access to network storage. The *hostile* model describes the case that credentials are required but are not available to the investigators.

A third aspect that can be used to classify forensic access to network storage are the circumstances of access. Here we distinguish access in the context of a live analysis during a search and seizure operation. While under time pressure and within a partly hostile environment, investigators usually operate within precisely defined legal bounds of criminal procedure. This is to be distinguished from access from within the controlled computing environment of a forensic lab, a context in which legal bounds increasingly blurr with the time since the search order was issued in many jurisdictions.

1.2. Related work

Research attention regarding the topic of "cloud forensics" has been increasing over the years. It can be roughly structured regarding the focus of evidence collection: on the client, over the network, or on the server (see Fig. 1).

The major part of previous work has been on the analysis of client devices that access cloud servers, as surveyed by Manral et al. [17]. More recently, focus shifted towards accessing cloud servers over the network using Web APIs [26, 27]. Download and synchronization tools like `rclone` [8] use these APIs but have deficiencies when used in a forensic context [3]. Roussev et al. [25] discuss experiences developing tools that access file-base cloud storage as well as more complex data objects like Google docs.

From looking at previous work, two areas appear less well studied (see Fig. 2): Firstly, there appears to be much potential to study more complex data objects such as those provided by Software-as-a-Service (SaaS) cloud deployments, an example being Google docs [25]. Simple file storage services such as those studied in this report are an instance of such deployments. Secondly, we are not aware of any

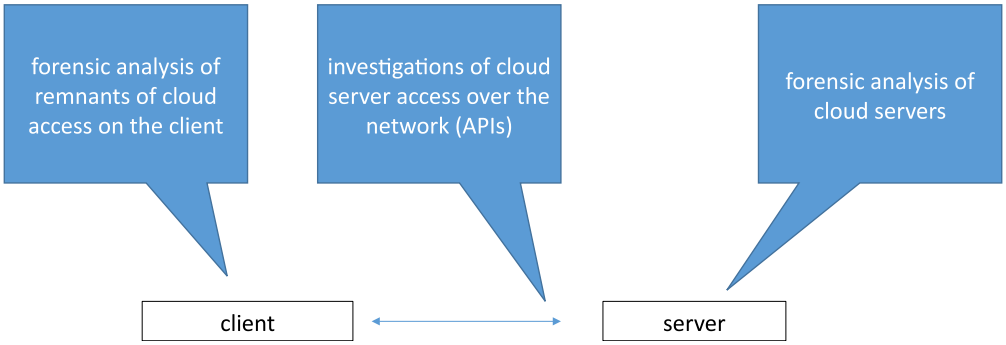


Figure 1: Areas of related work.

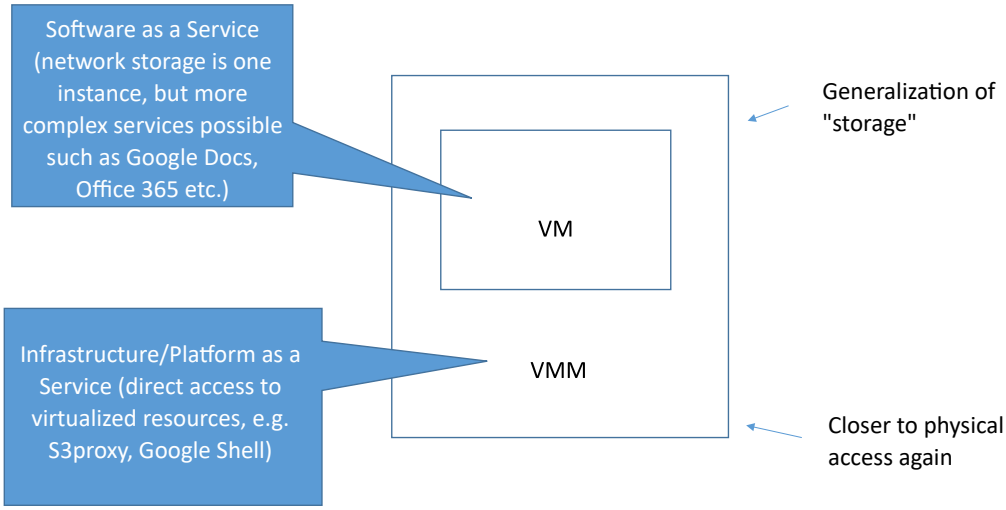


Figure 2: More general view on cloud computing.

work studying Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) cloud deployments. Since such environments offer direct access to virtualized resources (such as AWS S3proxy or Google shell), we expect that tools and methods for these environments will exhibit many similarities to work in classical disk analysis [4].

1.3. Context of this study

This document presents the results of a student project that was performed at the Department of Computer Science at Friedrich-Alexander-Universität Erlangen-Nürnberg in the winter term 2021/22 as part of a course on “Advanced Forensic Computing” (“Fortgeschrittene Forensische Informatik”). Six groups of students analyzed a total of six different technologies regarding their suitability as sources of digital evidence. The set of technologies was selected as follows:

- We began with an arguably representative set of the most relevant network storage technologies that are commonly available today, namely Dropbox, Google Drive, Microsoft’s OneDrive, and Apple’s iCloud.
- For historical reasons, with Samba/SMB we added a rather basic service that is however still widely used today.
- With Nextcloud we chose a technology that offers similar features as the commercial services above, but which is based on open-source software and can be self-hosted. Similar to Samba/SMB, this

gives a unique opportunity to control and investigate server-based behavior not accessible through a network interface.

The results of these analyses were integrated into this report. All students that contributed to the results are listed as co-authors of this document. Andreas Hammer and Felix Freiling integrated the individual project reports of the groups into this document. Linus Düsel supported the students in performing their project work and supported the final revision of this document. The original idea for this exercise came from Vassil Roussev, who attended and commented the project presentations and contributed to the conceptual discussions and final revisions. Since it was merely recommended and not mandatory that students submit their reports in English, some parts of this text are in German.

The task of the students was to analyze each of the network storage technologies according to two aspects that refer to critical aspects of forensic evidence collection, namely in what way does evidence collection interfere with the evidence, and how easy is it to prove the provenance of data in a forensic investigation. More specifically, the two aspects were formulated as follows:

1) (Interference) What effect does data acqusition by the client have on the data stored on the server? How does the degree of change depend on the type of access?
2) (Auditability) To what degree can access to data on the server at a particular time be proven at a later point in time? Does the technology support delayed verification of data acquisition in any way?

The circumstances of network access were as follows:

- Access model was *cooperative*, i.e., access credentials are available.
- Access was to be performed in a *lab context*.
- Legal questions regarding transnational data access could be excluded from consideration.

Groups consisted of 4–5 students each. The task of each group was to investigate one form of network storage regarding the above aspects and to create software that could acquire digital evidence from that network storage. The main authors of the groups are given for every chapter.

1.4. Roadmap

This report is structured as follows: Following this introductory chapter, the six student contributions (Samba, Nextcloud, Dropbox, Google Drive, OneDrive, iCloud) are presented as individual reports. The reports each consist of a platform introduction, description of findings and discussion of the exercise questions on interference and auditability. In the concluding discussion we compare the different technologies and develop a taxonomy of storage services that can be used to assess other cloud storage services with regarding the evidental value of data acquired from them.

2. Samba

2.1. Introduction

Server Message Block (SMB) is a protocol for communicating with file and printer services. It is one of the most common instances of a network file system, designed to read and write files on a remote host over a local area network. Since SMB is a request-response protocol, it always needs a transport protocol to operate. In the past, solely NetBios was used. Today it runs directly on top of Transmission Control Protocol on port 445 [16, p.5-6].

This section gives a brief introduction to SMB and presents the design and implementation of `smbclone`, a tool to clone a samba share for further forensic investigation. We report on the properties of the tool regarding Interference and Auditability, as well as the parameters that can be set regarding authentication between the client and the server.

2.2. Background

In the UNIX environment the most commonly used SMB server is Samba, which is an open source implementation of the SMB protocol. The implementation is included in most Linux distributions and when enabled, it will start automatically during the boot process. Setting up a server is pretty straightforward and the configuration can be completed with a single file. Additionally, it is possible to execute a logon script on the samba server and the group policy can be managed through `poledit`.

2.2.1. Authentication and Authorization. Samba needs to reconcile the security models of Unix and Windows systems. It must identify users by assigning them to valid user names and groups, authenticate them by checking their passwords, and then control their access to resources by comparing their access permissions to file and directory permissions [30]. Not only are there different operating systems to consider, but also the fact that Samba supports different user authentication mechanisms. Therefore the following section will treat the aspect of authentication mechanisms within the SMB protocol (Samba implementation), as well as different authorization stages of users. The corresponding configuration parameters that can be found in the `/etc/samba/smb.conf` will also be mentioned.

The Client-Server authentication is performed via the Generic Security Service Application Programming Interface (GSS-API) specification. This specification guarantees that messages are sent in an opaque way, as so called Tokens. Hence messages can be transmitted via insecure networks such as the internet as message security is guaranteed. The GSS-API is either implemented by the Kerberos or the NTLM authentication protocol. For Kerberos authentication the parameter `client use kerberos` in the configuration file can be modified. The default setting is `desired`, which means the authentication is tried and if it fails there is an automatical fallback to NTLM.

Since SMB version 2 passwords transmitted between the client and the server are encrypted by default. Nevertheless there are two configuration options that can still reverse that setting. Both settings are deprecated since Samba versions 4.11 and 4.13 and the "support for plaintext (as distinct from NTLM, NTLMv2 or Kerberos authentication) will be removed in a future Samba release"[29]. Up to the current 4.15 release they are still active though. The first parameter is `encrypt passwords`, which "controls whether encrypted passwords will be negotiated with the client"[29]. The second parameter is `client plaintext auth`, which "specifies whether a client should send a plaintext password if the server does not support encrypted passwords"[29].

Encryption mechanisms were introduced in SMB version 2 with the possibility to protect messages with Message Authentication Code signatures using cryptographically secure keys. A list of available signing algorithms which are available for negotiation and their order are set with the parameter `client smb3 signing algorithms` (or `client smb3 signing algorithms` on the server side) and default to the list `AES-128-GMAC, AES-128-CMAC, HMAC-SHA256`. From SMB 3.x on authenticated encryption and integrity protection are supported through the use of AES-128 and in version 3.1.1 through AES-256. The `client smb encrypt` parameter controls whether a client should try or is required to use SMB encryption. Likewise `server smb encrypt` controls whether a remote client is allowed or required to use SMB encryption. The `default` setting means, that negotiation of encryption is enabled, but data encryption is turned on neither globally nor per share. To check Encryption and Signing information of a SMB (Samba implementation) share mounted on linux the command `smbstatus --shares` can be used.

In the Samba configuration file there are various options that can be used to control access to shares, as listed in table 1. The scope for all options is a share. The `guest only` parameter is a boolean set to *no* by default. If it is manually set to *yes*, access to the corresponding share without password authentication, a global access, is allowed.

Option	Function
`admin users`	perform operations as root
`valid users`	connect to share
`invalid users`	denied access to share
`read list`	read-only access to writable share
`write list`	read/write access to readonly share
`max connections`	max number connections to share at given time
`guest only`	if yes: only guest access allowed
`guest account`	account used for guest access

Table 1: User access options

2.2.2. Protocol Messages. We now give a short explanation about what the protocol messages are used for. Afterwards we will explain the messages that were the most important for our research and show the structure of them in Wireshark.

The SMB2 Protocol uses messages for the communication between the server and the client. Request messages are used by the client while response messages are sent by the server. Each of these messages starts with a header of a length of 64 bytes [18]. It contains the operation code that has been requested by the client or responded by the server [18]. The header is followed by the actual request or response message. The content depends on the message type that has been sent.

- **SMB2 NEGOTIATE**

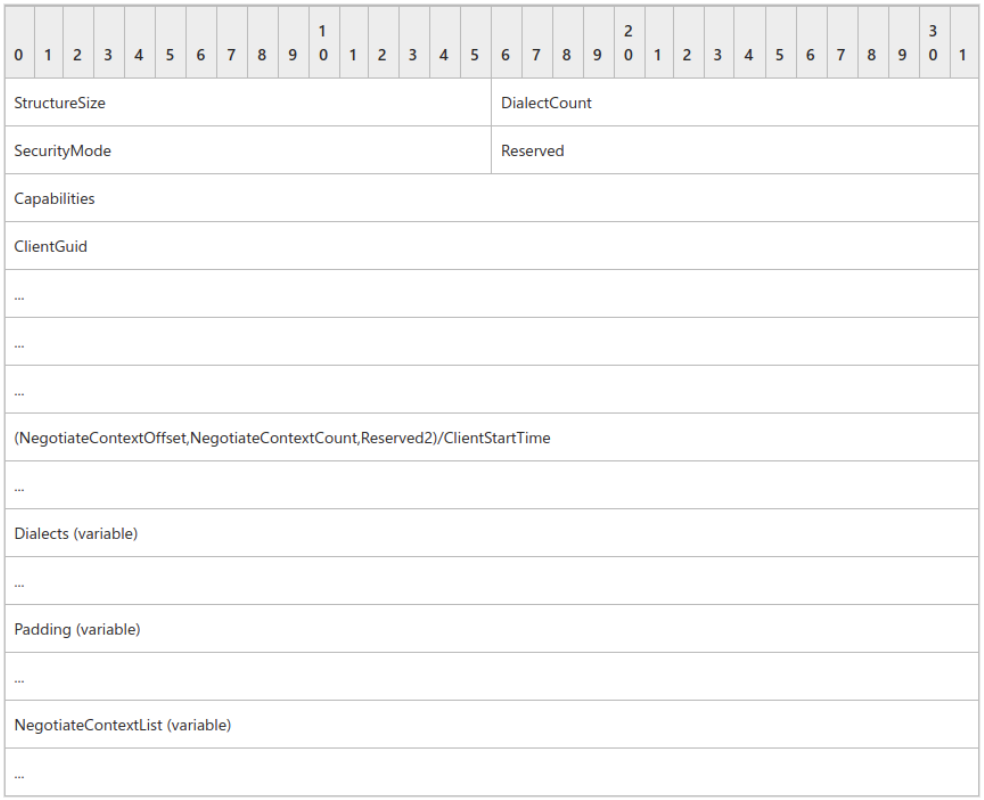


Figure 3: The structure of the SMB2 NEGOTIATE Request [18]

As in 3 shown, the NEGOTIATE request contains a *dialect variable*. Dialects are different versions of the protocol. Via that request the client stores the dialects it understands in an array of 16-bit

integers [18]. In doing so it chooses at least one out of the following dialects:

Value	Meaning
0x0202	SMB 2.0.2 dialect revision number.
0x0210	SMB 2.1 dialect revision number.
0x0300	SMB 3.0 dialect revision number.
0x0302	SMB 3.0.2 dialect revision number.
0x0311	SMB 3.1.1 dialect revision number.

Table 2: Dialects used by the protocol [18]

As a result on the request the server sends a response with the preferred dialect out of the suggested ones by the client back as can be seen in 4.

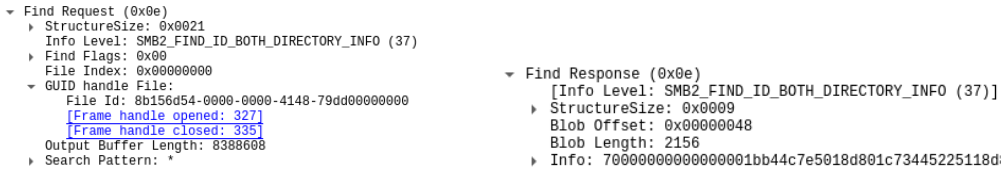


Figure 4: NEGOTIATE Request and Response in Wireshark

- **SMB2 QUERY_DIRECTORY**
  This message is sent for an open directory. Though it is stated as QUERY_DIRECTORY in the documentation, it can be only tracked as *Find Request and Response* in Wireshark. As one can see in 5, the *GUID handle File* is part of the message structure. The File Id is returned from a SMB2 CREATE Request and is used for the identification of the directory later on [18].

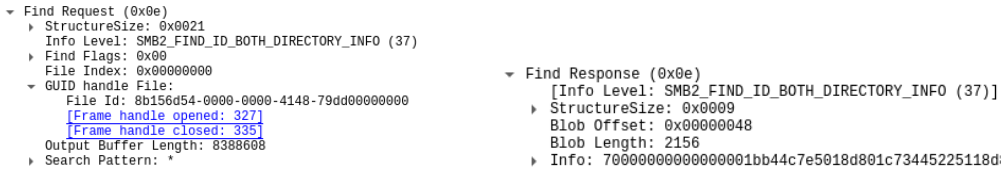


Figure 5: Find Request and Response in Wireshark

- **SMB2 CREATE** The SMB2 CREATE message is sent if a file is either created or opened [18]. As the request pictured in 6 shows, the file *foo.txt* comes up in the *filename* section. Looking at the CREATE response points out that the file already existed and was only opened as described in *Create Action*. Also, the before mentioned *GUID handle file* variable is used again to identify the file *foo.txt*.
- **SMB2 READ** The SMB2 READ message is sent if a file has been opened [18]. By sending a READ request the client can request a read operation on the file `foo.txt` stated by the *GUID handle File* [18].
- **SMB2 CLOSE** The SMB2 CLOSE message is sent if a file was closed that has previously been opened with a SMB2 CREATE request [18]. That can again be identified with the *GUID handle File*.

2.3. Tool

We now describe the design and implementation of our tool `smbclone`.

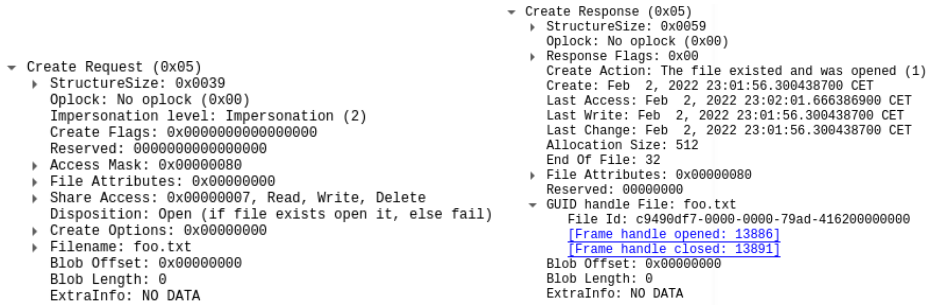


Figure 6: CREATE Request and Response in Wireshark

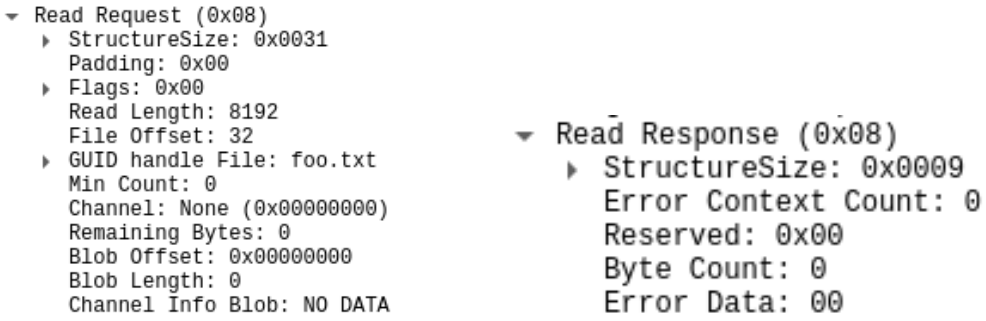


Figure 7: READ Request and Response in Wireshark

2.3.1. Approach. Our goal was to create a tool to forensically clone a SMB share via the command-line. This tool should be able to send SMB Messages to a specified server to iterate over the directories to mimic the directory structure and copy the remote's file contents and metadata. To send these messages we decided on writing the tool in Python 3 using the smbprotocol library [2]. This library claims to implement SMBv2 and SMBv3 based on the official MS-SMB documentation [18].

Simply cloning the directories to the local file system would result in a loss of metadata. This is due to limitations of different file systems. Some file systems for example do not differentiate between lowercase and uppercase in a path. Different file systems might also store different timestamps or store them with a different precision. To circumvent these limitations, the tool should also write all the metadata to a log file. This way no metadata should get lost through the limitations of attempting to clone the remote files and directories to the local file system.

2.3.2. Implementation. The Tool should be easy to use. For parsing arguments we used the argparse module of the python standard library [12]. The generated help page gives an overview over the needed arguments and optional flags (see Fig. 9).

The arguments that need to be specified are:

- `share` - To specifiy the remote share that should be cloned.
- `output` - To specify an output directory into which the remote share should be cloned into. This directory must not exist.
- `username` - The username needed to authenticate to the server.
- `password` - The password needed to authenticate to the server.

If all of those arguments are specified, the tool proceeds by iterating the contents of the root directory of the share. This is done recursively on all subdirectories. Endless loops introduced for example by symlinks to a parent directory are avoided by memorizing a set of the inode numbers of searched directories and checking the inode number of the current directory against that set. For each directory found on the

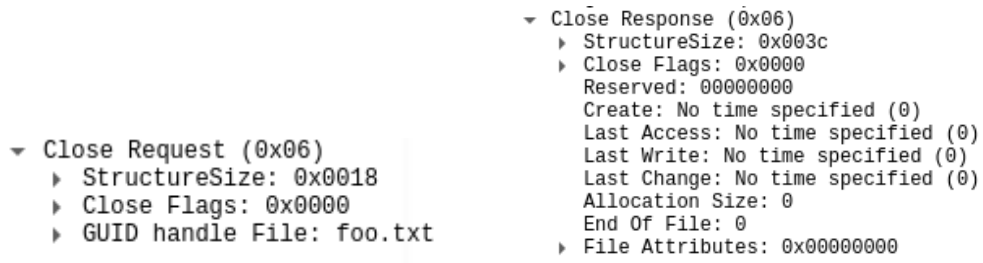


Figure 8: CLOSE Request and Response in Wireshark

```
$ smbclone --help
usage: smbclone.py [-h] [--verbose] [--md5] [--sha256] [--sha512]
                   [--port PORT]
                   share output username password

Clone a smb network share.

positional arguments:
  share                 The network share that should be cloned (e.g.
                        "//localhost/userdirectory").
  output                The output directory that the share should be cloned
                        to (has to be empty or non-existent).
  username              The username that should be used.
  password              The password that should be used.

options:
  -h, --help            show this help message and exit
  --verbose, -v         Set verbosity.
  --md5                 Calculate md5 sums for files.
  --sha256              Calculate sha256 sums for files.
  --sha512              Calculate sha512 sums for files.
  --port PORT, -p PORT
```

Figure 9: smbclone help message

remote a corresponding local directory is created to mimic the remote directory structure. Also for each remote file a local copy is created. The modified timestamp and access timestamp of the local copy are set to mirror the remote share. The tool also detects hardlinks on the remote share by comparing the inode numbers and mirrors them on the local file system.

The directory structure is traversed using depth-first search. This is not done by design, but simply because of the recursive nature of our implementation. We note this, because it can impact the directory structure of the local copy when a known directory (symlink) is not traversed further.

All of this is possible by using the smbprotocol library [2]. To iterate over a directory we used the function `smbclient.scandir()`. It yields a generator that generates `smbprotocol.SMBDirEntry` objects. Those include information about the type of entry (directory, file(, symlink - not for Samba on Linux)). SMBDirEntry also defines a function `stat()` which returns a `smbprotocol.SMBStatResult` object containing metadata about the entry. `smbclient.stat()` can also be called for any provided

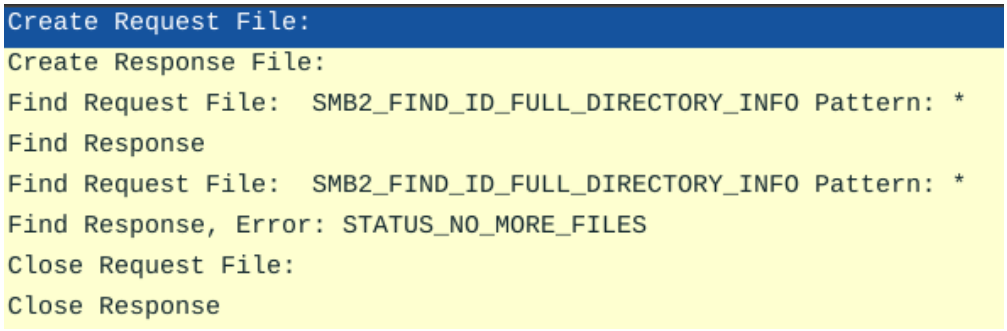


Figure 10: Wireshark Logs when calling `smbclient.scandir()`.

path. To open and read a file we use the function `smbclient.open_file()` and `read()` on the returned Reader object.

While verifying the readonly capabilities of smbclone we found that smbprotocol requests write and delete access in some cases, just in case the developer wants to call a function that would need these access rights.

An example of this is `smbclient.scandir()`. The `CREATE` request requests write and delete capabilities. We therefore decided to fork smbprotocol and call our fork smbprotocol-readonly [9]. This fork only ever requests readonly access. We achieved this by changing the requested access parameters to `"r"` instead of `"rwd"`. We also nulled the bitmasks for write and delete, just in case we overlooked an instance of the library requesting write or delete access.

2.3.3. Verification on the client-side. In this section the readonly access of `smbclone` is verified on the client-side. To verify the readonly capabilities on the client side we called the functions we used in smbclone interactively in the python interpreter.

Before executing the lines in the following, we first provided `smbclient` with the authentication details like so:

```
>>> smbclient.ClientConfig(username='sambauser', password='sambapass')
```

- **scandir**
  We iterated over the result of scandir by executing the following line in the python interpreter:
  ```
  >>> list(smbclient.scandir("\\\\server-ip\\sharename"))
  ```
  The `list()` function is used to iterate over the generator returned by `smbclient.scandir()`. As the line is interpreted wireshark logs the sent messages as shown in figure 10.
  In the `CREATE` request's field "Access Mask" only the Read, Read EA (Extended Attributes) and Read Attributes bits are set (see figure 11). This differs from the behavior of the original smbprotocol library, which also requests write and delete access.
- **stat**
  When interpreting the line:
  ```
  >>> smbclient.stat("\\\\server-ip\\sharename\\somefile.txt")
  ```
  Wireshark again logs a CREATE request. The "Access Mask" field only has the "Read Attributes" field set.
- **open_file + read**
  Opening (and reading) a file with the following command:
  ```
  >>> with smbclient.open_file('\\\\server-ip\\sharename\\file.txt') as f:
  ...     f.read()
  ```

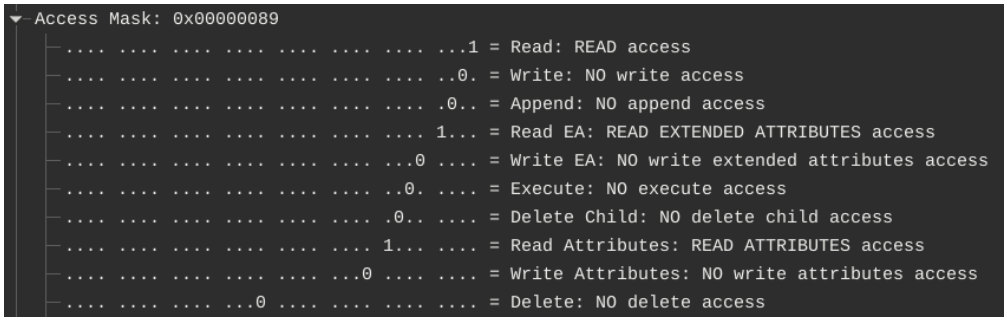


Figure 11: Wireshark Access Mask Field in CREATE Request for `smbclient.scandir()`.

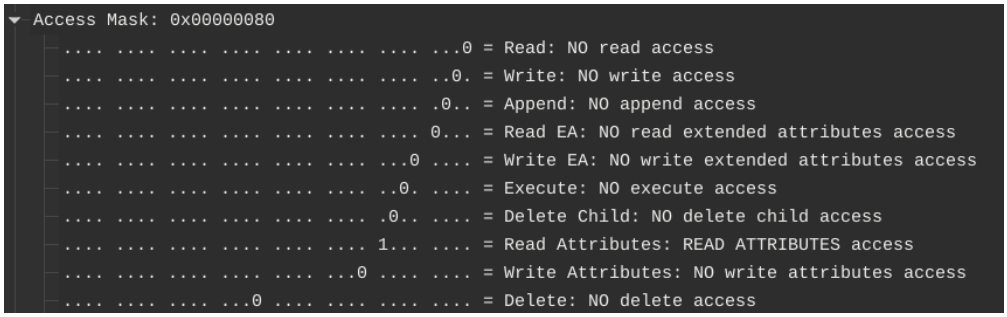


Figure 12: Wireshark Access Mask Field in CREATE Request for `smbclient.stat()`.

sends a CREATE request followed by a READ request. To be exact only the `smbclient.open_file()` function send the CREATE request. The CREATE request's "Access Mask" field has the bits set for "Read", "Read EA" and "Read Attributes".

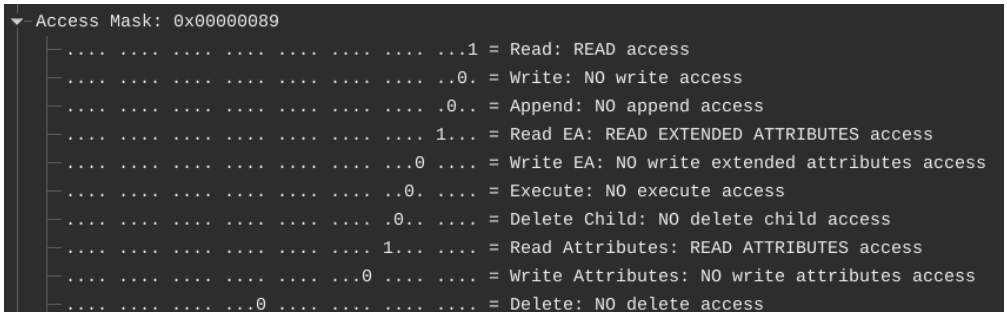


Figure 13: Wireshark Access Mask Field in CREATE Request for `smbclient.open_file()`.

2.3.4. Verification on the server-side. While it is important the smbclone tool does not send messages that instruct the server to modify its contents, which we verified in section 2.3.3, in this section we verify that the share's contents are actually unchanged after calling the tool.

To verify the directory structure and the file contents are unchanged, the diff tool can be used on the server.

We set up the server directory by copying example files and directories into the shared directory. We then ran smbclone and called diff to recursively find differences between the shared directory and the directory containing the example files. It could not find any differences. Figure 14 presents how this could be done in a normal linux shell.

```
$ rm -rf share_dir
$ cp {a example_files/ share_dir/
# ... smbclone ...
$ diff {r share_dir example_files
$ echo $?
# ... No differences! ...
```

Figure 14: Linux Shell Commands to Verify File Content and Directory Structure Integrity.

```
$ rm -rf share_dir
$ cp {a example_files share_dir
$ find share_dir {exec stat {} \; > metadata_list.before
# ... smbclone ...
$ find share_dir {exec stat {} \; > metadata_list.after
$ diff metadata_list.before metadata_list.after
# ... varying results ...
```

Figure 15: Linux Shell Commands to Verify Metadata Integrity.

While the file contents and the directory structure are unaffected, the metadata of the files and directories might still be affected by smbclone. To test the effects of smbclone on the metadata we executed the commands like shown in figure 15.

The command `find <dir> -exec stat {} \; > <output_file>` calls stat on every file and directory in `<dir>` (recursively) and writes the output to `<output_file>`.

The metadata, specifically the access timestamp, might differ, as even calling stat on a file can change the access timestamp. Wether the access timestamp is modified depends on the access timestamp policy used when the disk the folder is located on was mounted. On most modern linux distributions the default mount option is “relatime”. This means that the access timestamp is only updated, if the modify or changed timestamp is older than the access timestamp.

2.4. Summary

With the implemented smbclone tool, it is possible to mirror all files from a source folder, with a SMB server as source into any specified target folder, without altering any of the folders contents. Furthermore, we showed that using the tool the metadata of the mirrored content is not altered by the cloning process. This means that the creation timestamps stayed the same, so that it looks like the files were not moved at all.

3. Nextcloud

3.1. Introduction

One way to enjoy the benefits of the cloud without having to rely on commercial providers is self-hosting. Based on open source software, Nextcloud is a technology to establish self-administered cloud storage reducing the effort that is traditionally associated with hosting a cloud. Nextcloud offers various options to access data stored in a Nextcloud instance. We first give some background on Nextcloud

and the used scenario in Section 3.2. Then we review the access options that were used to access data and metadata alike in Section 3.3. Particular focus is put on a custom tool, which is intended to make the data acquisition process as simple as possible as well as similar to accessing a normal hard drive (Section 3.4). Finally, we summarize by discussing problems and limitations regarding the extraction of data from Nextcloud instances.

3.2. Background

3.2.1. Benefits of using Nextcloud. Many established providers (for instance Microsoft OneDrive, Google Drive, Apple iCloud or Dropbox) offer customers the ability to store data on their servers without being transparent where and how the data is stored or which actions are performed on the stored data in the background. If the user wishes to have more granular control of these factors, these solutions are usually not an option. In stark contrast, Nextcloud is a free open source software which enables the user to easily maintain or host a cloud instance and offers access and management of the underlying file system. Such a self hosted cloud solution is configurable and extensible to a wide variety of needs while preserving full control over personal or confidential data. Depending on the scenario (resident hosting or allowing external providers to host the instance), both Infrastructure-, Platform- and Software as a service scenarios in the Nextcloud context are possible, while the most common scenario is usually ”SaaS”-like.

During the investigations described in this report, the interaction with the chosen Nextcloud instance felt somewhat similar to a ”remote file system”, while any further interaction with the server was not necessary. This aspect is elaborated upon in more detail in Sections 3.5 and 3.6.

3.2.2. Used infrastructure. In order to be able to perform our analysis, it was necessary to set up a Nextcloud instance for testing purposes. Since we hosted the Nextcloud instance ourselves, we had the possibility to look at the file system on the server, establish a ground truth reference and eliminate any interference from outside factors. To enable reproducibility we detail the setup used by us in the following paragraph.

A Nextcloud instance demands a so called LAMP stack[1]. This acronym refers to a stack consisting of open source software that is usually installed cohesively to allow a server to host dynamic websites and web apps. The basis for this stack is a **L**inux distribution as the operating system of the server, in this particular case Ubuntu 20.04.3 LTS was used which was installed on a virtual machine that was running on top of a Proxmox Virtual Environment 6.4-13 hypervisor. For the actual web presence of the cloud a web server is needed. In our opinion **A**pache HTTP Server 2.4.41 was chosen for this purpose, as it is used in a wide variety of environements and relatively well documented for forensic purposes. For the backend a relational database system is required, for instance **M**ySQL or **M**ariaDB, the latter used by us in version 10.3.32. The final part required is a programming language. For Nextcloud it is recommended to use **P**HP, which’s version 8.0.13 was used by us. Other feasible alternatives would have been **P**erl or **P**ython. Having fulfilled these prequesites we downloaded the Nextcloud version 22.2.3 files and configured them to this setup. Additionally, in order to enable a secure connection to the cloud a certificate was obtained using certbot / letsencrypt 0.40.0. Furthermore a reverse proxy had to be configured to forward requests on the given subdomain to the corresponding virtual machine.

3.2.3. Setup. In the following chapter the simulated and investigated user behaviour scenario is explained to establish an entry point for the following analysis.

Since the considered cloud setup was resident hosted and therefore the entire user privilege hierarchy was easily comprehensible. The evaluated setting consists of ten concurrent users with varying privileges,

1. https://docs.nextcloud.com/server/latest/admin_manual/installation/source_installation.html

namely five admin accounts and five standard users. Despite Nextcloud being extensible with numerous additional plug-ins or add-ons (e.g. Collabora Online, Calendar, Mail capabilities etc.), the focus of this report was defined to be a stock configuration without any other modules installed in order to keep the investigated scenario as close to the most likely encountered scenario in a real setting. This inevitably leads to some simplifications in most of the multi-user interactions, particularly with shared resources like folders. In this case one user grants other users access to a folder which is conesquently marked as shared. This concurrent use model is especially interesting from a forensic perspective, as it might entail unforseen behavior.

The common user behaviour was simulated (e.g. downloading, changing, tagging, favoring and deleting of files) to check if and how these interactions are traceable in a later analysis.

Since the self hosted cloud scenario allowed us to access the server directly, a variation of the proposed scenario was also investigated, this time including SSH or local access. By doing this, it is possible to get a reference to serve as ground truth when evaluating the interference issue described in the problem description. In our concrete scenario we chose the access via remote SSH, which allowed us to gain insight into changes made to the underlying file-system or to verify that relevant actions are persistent in the internal database during the investigation.

3.3. Review of various cloud access methods

In this section a summary of our findings in terms of data gained from accessing a Nextcloud instance over other methods than the web interface is given.
The only method of accessing Nextcloud besides via a browser is its WebDAV-API. We therefore restrict our findings to this interface for the time being, discussing the effects of mounting the Cloud as a network drive, utilizing the Nextcloud proprietary client and present our own Python based custom forensic CLI-tool. After we exhausted the possibilities of the WebDAV interface we compare the information gained with the data accessible over SSH, including the interference of the other access methods and their auditability.

3.3.1. Mounting Nextcloud as a network drive. All common operating systems offer the mounting of Nextcloud storage as a ”network drive”.[2] To achieve this the WebDAV protocol is used to connect and synchronize files. In contrast to the proprietary Nextcloud Client (further discussed in section (3.3.2)), this method is independent of additional software due to direct communication of the file explorer with the API. This allows the user to have an effortless ”drag&drop-like” experience when interacting with the cloud content similar to the local file system. While this significantly improves the ease of use, the possibilities of investigating the Cloud content over this instance is rather limited. Since there is no sophisticated meta-information acquirable this way other than some timestamps (e.g. for the time of creation), which is also available using the methods discussed in the following sections, not much attention was given to this access option.

3.3.2. Proprietary Nextcloud Client. Another way to access data in Nextcloud is through Nextcloud client. It is available for most operating systems like Windows, macOS and various Linux distributions and allows the user to select directories on your local machine that are automatically synchronized with the Nextcloud server. Just as with the network drive mounting method it utilizes the WebDAV interface of the Nextcloud instance.

Forensically, an analysis of this client is interesting in two ways: On the one hand, this client could be used to forensically secure the data stored in the cloud. Hence, it must be checked whether and to what

2. https://docs.nextcloud.com/server/latest/user_manual/en/files/access_webdav.html

extent metadata is transferred. On the other hand, it is also possible that this client is already installed on a computer that is to be examined. For that reason, it has to be analyzed, which data is created and stored by the client in addition to the user data.

Regarding the transferred metadata, it was unfortunately discovered that the synchronization causes some problems concerning timestamps. When synchronizing data from the server to the client, timestamps behave as described below:
The access time and modified time are inherited from the server. The changed time is adapted locally to the time of synchronization. Likewise the access time on the server is also updated.

For the return direction, i.e. a synchronization from client to server, the time stamps behave correspondingly.

Looking at the data that the client creates in addition to the user data, it was observed that not much forensically relevant information can be obtained. The only data of interest are the XXX_sync.log files and the .sync_XXX.db files which are created for each folder to be synchronized (XXX used as a placeholder for the corresponding folder name). The XXX_sync.log files contain an entry for each synchronization activity that has happened for the corresponding folder. This information includes a timestamp, the duration of the synchronization activity, the affected files, the instruction (e.g. upload or download), the modification time and some other attributes that are forensically not as relevant. Besides these log files, there is also a .sync_XXX.db file for each folder. This database contains the last time of synchronization and information about each file in the folder. This information covers data about the path of the file, the inode number, the modification time, the MD5 hash, the filesize and remote permissions on that file on the server. Although there is still more information in that database, these attributes were the most valuable in a forensic context.

3.4. Engineering a custom forensic CLI-tool

3.4.1. General Requirements. As mentioned before, interference has to be kept to a minimum as not to alter the reviewed storage or change relevant metadata unnecessarily. To ensure this, a solution was developed that focuses purely on downloading, hashing and reviewing data. When challenged, an analyst has to be able to present the undertaken steps of an analysis in an easily reviewable manner; thus utmost attention has been dedicated to logging functionalities. Furthermore the tool should aim to be as lightweight and user-friendly as possible to ensure ease of use.

3.4.2. Technical requirements.

- python 3.10
- certifi version: 2021.10.8
- charset-normalizer version: 2.0.10
- idna version: 3.3
- lxml version: 4.7.1
- python-dateutil version: 2.8.2
- requests version: 2.27.1
- six version: 1.16.0
- tabulate version: 0.8.9
- urllib3 version: 1.26.8
- variables version: 0.0.1
- webdavclient3 version: 3.14.6
- paramiko version: 2.9.2

3.4.3. Features. Due to their striking prevalence in the Nextcloud software ecosystem, a WebDAV API based solution was opted for. Several Python libraries were tested, in the end ”webdavclient3” was chosen due to its clean code structure, lightweight implementation and robustness. The main features of the custom tool include – but are not limited to – storing your cloud access credentials in a separate file to prevent overhead, navigating and accessing the cloud storage in a shell-like manner using commands like *ls / dir, cd, pwd, etc.*, downloading files and calculating hashes to track changes. Most people using forensic tools are probably familiar with these commands, using the custom tool will thus be as seamless as possible. No matter the command, every call issued to the API will be logged with a timestamp, a username and the response. This strongly enforces transparency and auditability in a forensic analysis. The tool itself is also highly customizable and extensible due to the modular and readable structure of the underlying code.

3.5. Server-side access capabilities via SSH connection

Regarding the point of interference, it was mandatory to check if there are any inconsistencies between accessing the data via WebDAV or via direct access on the server. Since the majority of the project group had no physical access to the cloud server during the whole time of investigation to simulate the aimed for external remote lab access scenario described before, this was achieved via SSH connection. As specified in section (3.2.2) the cloud server is running Ubuntu, which enables linux shell commands for file system traversal.

Thus accessing the actual data stored on the server is possible by entering the directory ”`/mnt/hdd/`”. This directory contains one sub-directory for every user registered in this cloud instance.

```
ffi_user@ffi-cloud:/$ cd mnt/hdd
ffi_user@ffi-cloud:/mnt/hdd$ ls
Lukas  UnixChris  __groupfolders  admin  appdata_oc5z3d4qtfa7  chris  index.html  kaiesser  nextcloud.log  nicomueller  oli  owllilicious  updater-oc5z3d4qtfa7  updater.log
```

Figure 16: Folders for every user on the Nextcloud-instance

```
ffi_user@ffi-cloud:/mnt/hdd/oli/files$ ls
 Documents   Gemeinsam  'Nextcloud Manual.pdf'  'Nextcloud intro.mp4'   Nextcloud.png   Photos   Readme.md  'Reasons to use Nextcloud.pdf'   Talk   Vorlagen   test.odt
```

Figure 17: Files stored in user account on Nextcloud-instance

This is illustrated in Figure 16. By switching into ”`/mnt/hdd/USER/files`” the analyst can access the actual data stored in the cloud account of the corresponding user. Aforementioned description is illustrated in Figure 17. That directory provides an important entry point in terms of interference, since it allows to compute the sha256hash-sum of every file on the server by using the `sha256sum` command. Now one is able to compare these hashes to the ones computed from the data downloaded by the forensic tool mentioned in Section 3.4 to answer the question if accessing the data this way leads to changes in the data. This procedure was automated by a python tool using `paramiko`. By doing this, it could be verified that the hashes of the files stored on the cloud server and the corresponding ones downloaded by the forensic tool mentioned before did not differ and therefore it can be be proven that there is little to no change to the data using the own forensic tool.

However, it is worth mentioning that the procedure described before has some limitations due to shared folders being stored only for the user creating the folder, while other invited users only get mapped to this directory by the server. Despite having access to the shared folder content in the forensic tool for every legitimate user, one is unable to find it in their directories on the server.

Additionally, if a specialist in computer forensic has access via SSH to the Nextcloud server then one is able to retrieve the forensically most relevant internal Nextcloud database including the table ”oc_activity”. An excerpt of this database is illustrated in figure (18). To find it one has to access

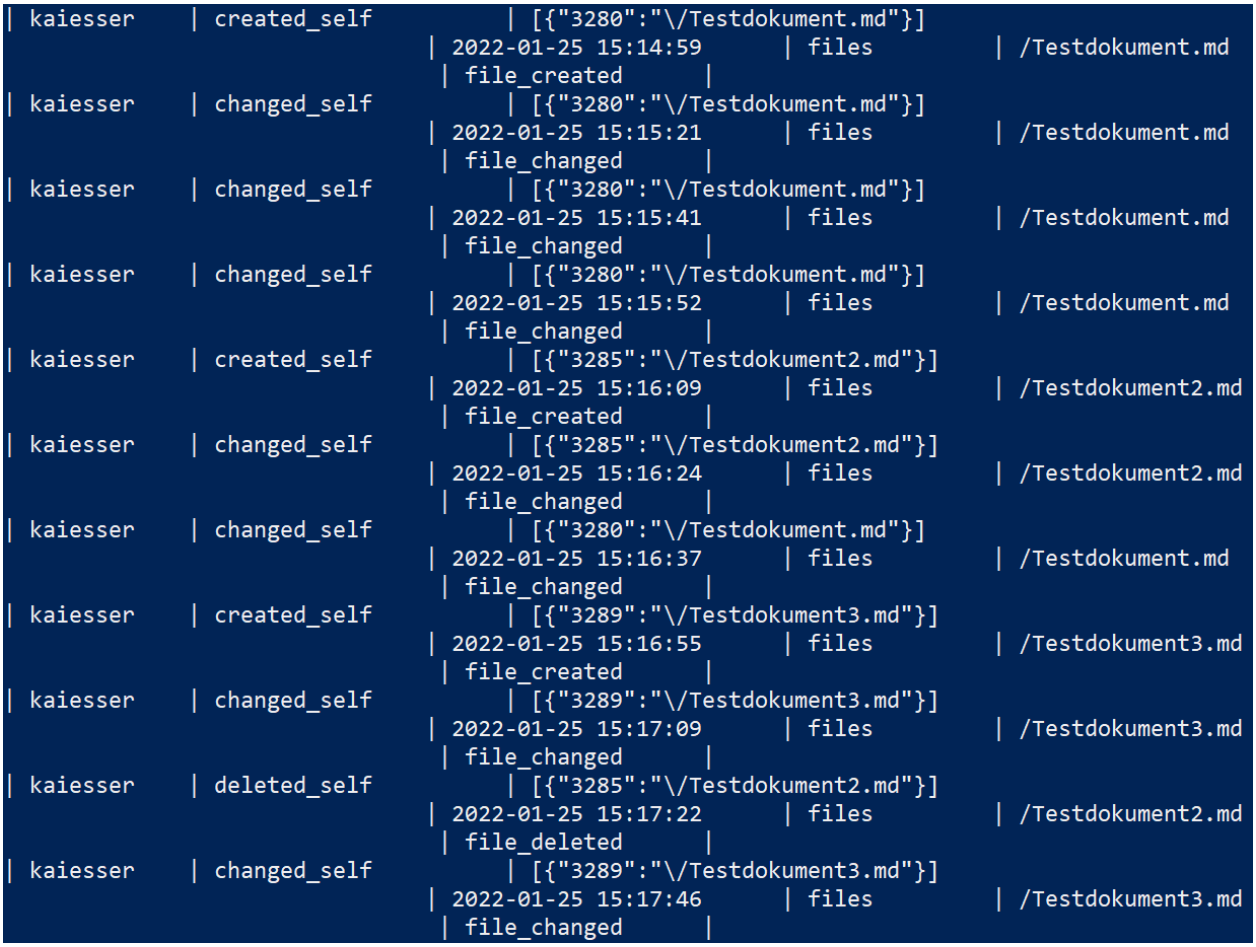


Figure 18: The forensically most relevant table of the internal Nexcloud database: excerpt of oc_activity

the file `/var/www/nextcloud/config/config.php`: in the tag ”dbtype” one can see which database is used by this Nextcloud instance. Alternatively, the other tags ”dbhost”, ”dbport”, ”dbuser” and ”dbpassword” can be used to connect directly to the database management system. In this database, a multitude of forensically relevant inforamtion is stored. Metadata about each activity of each user like timestamps from creation, changed or deleted is documented.

3.6. Limitations

During the analysis of the different access options we encountered some problems and limitations. These will now be described in detail.

One major issue is the lack of variety with respect to the interfaces offered by Nextcloud. In fact, the only option – besides the web-overlay – is the use of WebDAV. Therefore, regardless of the specific access method, only data can be accessed, that is provided by Nextcloud through this interface.

Speaking of data, the amount of metadata that Nextcloud shares is quite limited. Thus, only little further information is provided in addition to the actual user data. Apart from timestamps, which are, however, modified by the forensic access via WebDAV (as described in 3.3.2), no other metadata is shared.

The data of particular interest from a forensic perspective is stored in the internal database which cannot be accessed externally. In this analysis, consequent use of SSH access to the Nextcloud server

had to be made in order to be able to evaluate this database. Since Nextcloud is usually only used in small groups or even alone, there is a chance that the credentials to the server can be obtained, but it cannot certainly be taken for granted. Furthermore, the database itself is often protected by another password, which would also have to be obtained and is only likely to be accessible in a cooperative scenario.

Versioning is also limited to a certain extent. It only takes place on the server side. The version control uses a simple pattern[3] to check whether to keep the version of an individual file. Those versions can be restored by using the web-overlay but also via the WebDAV interface. Locally on computers where e.g. the Netxcloud client is installed, no versioning is done.

The last problem encountered is the way that Nextcloud handles shared folders: If person A shares a folder with person B, the actual data is still stored in the directory of person A. To allow person B to access the data, an entry is added to the Nextcloud database that states the access rights of person B. For this reason the server-side hashing function in our tool does not work properly on such shared folders. If the function is invoked on the data of person B, the shared files are not existent in the corresponding user directory, leading to problems. To include these files, our tool would have to look up the database for all entries affecting person B, which it is unable to do.

3.7. Summary

Only very limited information can be extracted from Nextcloud using its web- or WebDAV-based access methods. This is not surprising seeing how one of the main selling points for Nextcloud is its privacy in comparison to other services. This is especially problematic in a non-cooperating or hostile environment, which were outside of the scope of this report. Especially the lack of redundancy in terms of a lot of the metadata, which is exclusively stored inside the single database, can make forensic investigations particularly difficult, seeing how this presents a single point-of-failure. Given SSH or physical access, the database can be extracted and a proper investigation can take place. Without it only the most basic information can be derived, e.g. to which files the user has access or when the file was created and whether it is shared or not.

Interference with the data was not found to be an issue for any of the access methods, with all tested files no deviation of hash sums between local and the file in the cloud was found. Only metadata has been found to get changed during read access.

Nextcloud offers limited file versioning, depending on the total available user space. The worst granularity was found to be one version per week, provided that the data quota does not restrict versioning.

4. Dropbox

4.1. Introduction

In this section we analysed the Dropbox Cloud Service in terms of auditability and interference. As a result, we wrote functionality in Python that uses the Dropbox Python SDK to retrieve the current metadata state of all the Dropbox's content and saves it in structured JSON format, to allow further analysis. In addition we were able to fully download even deleted content on a user's Dropbox without interfering with its state meaning that we neither change anything within the content nor the metadata. Anyway there were some limits in terms of auditability as some information is only temporarily available like a file's revision history, which won't allow to restore all the revision ever existed.

On top of a basic account for the Dropbox we used an free trial of an elevated account to examine the information on the Dropbox Team functionality also using the Python SDK. The results show, amongst

3. https://docs.nextcloud.com/server/latest/user_manual/en/files/version_control.html

others, that with full permissions an admin of a team is able to retrace every action any member has taken on a team's folder, as these actions are logged, but also our application's actions are traced leading to higher interference.

In Section 4.2.1, we examine the existing Python Dropbox SDK that we used to develop a tool for forensically analysing the data in the Dropbox cloud. We register the Dropbox App in Section 4.2.2, and analyse the basic account in Section 4.2.3. Our tool is similar to established digital forensics tools and is easy and intuitive to use. The tool searches the stored files in the cloud including their metadata. The files can be downloaded without modification. Even deleted data can be recovered. In Section 4.2.4, we also examined the additional possibilities for analyzing Dropbox business team accounts.

In the following, the development and functionality of the tool will be explained in more detail.

We successfully developed a Python script that uses the Dropbox Python SDK to access the content on a user's Dropbox and furthermore export all current metadata of this content to the local system. We used a JSON format to save the metadata of the user's account and of all the files, folders and deleted metadata, we are able to access with our application.

In addition we used proper functions of the SDK to download all the content of the Dropbox including all revisions, that we are able to access, onto the local system, where the Python application is running.

It turned out that interference was not a big issue on file level, as the metadata and the content will not change, if we export all this information.

In terms of auditability it looks a bit different, as we encountered some limits, for example when accessing older versions/revisions of a file, we were only able to list up to 100 revisions and also restoring these revisions was time-limited, depending on our Dropbox account type.

In the second part of our analysis we wrote an additional script that retrieves information with an elevated account and the additional functions that come with it. We exported the result of the most relevant, most interesting functions of the SDK regarding `team` functionality into JSON format.

4.2. Analysis

We now describe the procedure of our analysis and present the results.

4.2.1. The Dropbox API. *"Getting up and running on the DBX Platform is fast and easy. The powerful, yet simple, API allows you to manage and control content programmatically and extend Dropbox capabilities in new and powerful ways."* [11]

Dropbox offers its API v2 with a set of HTTP endpoints. It uses HTTP POST requests with JSON formatted data. Applications can authenticate via OAuth 2.0. [10] There are officially supported Dropbox SDKs that function as wrappers for communication with the endpoints. We use the Python SDK to access the data on the Dropbox Cloud.

The Dropbox service offers the account types Basic and Business. The additional functions and features of a business account can only be accessed after payment.

All dates in the API use UTC and are strings in the ISO 8601 "combined date and time representation" format: `2015-05-15T15:50:38Z`.

4.2.2. Registering a Dropbox App. To use `Dropbox for Python`[4] we register an Application on the Dropbox website with our credentials.

Next to the settings like the application name, we can configure adequate permissions for pure purposes there. For this analysis we gave the application full permissions of all types, these are:

- Account Info: Permissions that allow your app to view and manage Dropbox account info

4. https://dropbox-sdk-python.readthedocs.io/en/latest/index.html

- Files and folders: Permissions that allow your app to view and manage files and folders
- Collaboration: Permissions that allow your app to view and manage sharing and collaboration settings

The `OAuth2` token we can generate has no expiration. In the next sections we use the this token in our python application to communicate with the endpoints.

4.2.3. Basic Account Analysis. In this section, we will focus on the free basic Dropbox account plan. As mentioned we generate a OAuth2 token in the Dropbox application settings to use the Python Dropbox SDK. This token is then used to initialize the Dropbox client session with `dbx = dropbox.Dropbox(OAUTH2_TOKEN)`. The returning object `dbx` can then be used to make requests to the Dropbox API. The methods of this provided object are meant to act on the corresponding user's Dropbox.

4.2.3.1. Account objects. There are two different types of account information objects, "FullAccount" and "BasicAccount".

We retrieve full information about the current user's account via `full_account = dbx.users_get_current_account()`. This returns an object of the class `FullAccount` and has some interesting fields:

- `account_id`: the user's unique Dropbox ID, i.e. dbid:AACQ9w1... (32 chars)
- `account_type`: what type of account this user has, i.e. `basic` or `business`
- `country` and `locale`: the country and language the user specified
- `name`: is an object containing amongst others surname and familiar name
- `email` and `email_verfied`: the user's email and if it is verified
- `team` and `team_member_id`: info about the team the user is a member of and it's unique member id if the user is part of a team

It is also possible to retrieve any user's account information by specifying his/her `account_id`. This can be done with `basic_account = dbx.users_get_account(<account_id>)`. This function returns an object of the class `BasicAccount` which has less fields probably due to privacy reasons, i.e. no information about `account_type`, `country`, `locale` or the `team` the user is member of.

4.2.3.2. Metadata. Every file and folder in Dropbox has metadata. There are three different types of metadata: **FileMetadata**, **FolderMetadata** and **DeletedMetadata**. The differences and similarities will be pointed out in the following section.

Fortunately there is a comfortable way to retrieve all metadata available of a user's Dropbox recursively starting at the root path, which is the empty path string. We can even include deleted content in this listing.

```
all_metadata = dbx.files_list_folder("", recursive=True,
include_deleted=True, include_media_info=True,
include_mounted_folders=True,
include_non_downloadable_files=True,
include_has_explicit_shared_members=True).entries
```

The result will be a list of metadata objects. All types of metadata provide the information about the path within the user's Dropbox and a name of the file or folder the metadata refers to. All files and folders have a unique ID.

We added functionality in our Python script to transfer the whole metadata into JSON format which enabled a easier analysis of the fields and debugging.

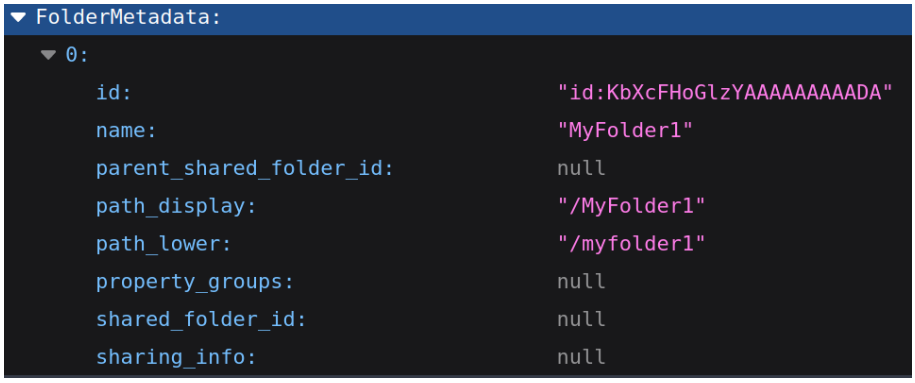


Figure 19: fields of the FolderMetadata object

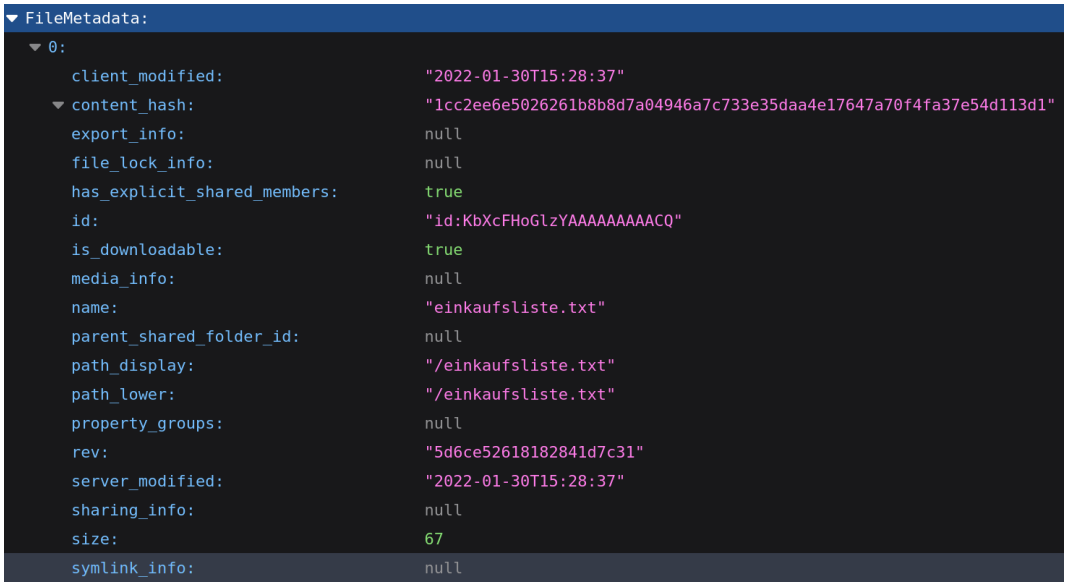


Figure 20: fields of the FileMetadata object

In Figure 19 we can see the provided fields for a object of type `FolderMetadata`.

We can see that there are no timestamps given. In Figure 20 we can see a `FileMetadata`'s fields providing quite more information that the metadata from a folder.

There are two timestamps: `client_modified` and `server_modified`.

The `client_modified` is the modification time set by the desktop client when the filed was added to Dropbox. Since this time is not verified, this should only be used for display purposes and not to determine if a file was changed or not.

The `server_modified`, on the other, hand holds the modification date when the file was changed on Dropbox.

The `content_hash` field can be used to verify that a downloaded file was neither corrupted nor altered in the process. While its content initially looks like a plain SHA256 checksum, it is actually computed as follows:

1) Split file into 4 MiB (4 * 1024 * 1024 Byte) chunks (the last chunk may be smaller)
2) Compute the SHA256 checksum for each chunk and concatenate their binary representations

3) `content_hash` is the hex representation of the checksum of the concatenation

While `content_hash` is helpful to get a forensically sound download of each file in the Dropbox account, it conflicts with the traditional approach to take a checksum of each file to guarantee it is not altered after being persisted. When our script downloads all contents of a Dropbox, it therefore also generates a file with the ending `.sha256sum`, which contains the usual SHA256 checksum for each downloaded file, and can be given as input to other tools, e.g. `sha256sum -c`.

We also see a field called `rev`. This is the unique identifier for the current revision of a file. This field is the same rev as elsewhere in the API and can be used to detect changes and avoid conflicts. Each change of the content leads to a new revision of that file and therefore to a new revision number. Each file has at least one revision since uploaded to Dropbox where the first one is the file's original version.

The revision number can be used to restore a certain state of a file. We will focus on this more precisely in the next section.

When a file or folder gets **deleted**, the current metadata object of a file or folder is turned into `DeletedMetadata`. That also means that content once uploaded to the Dropbox is persistently accessible even if deleted.

Anyway, functionality of permanent deletion is available, but only possible with `business` accounts.

In Figure 20 we also can see that the boolean field `has_explicit_shared_members` is set to true. This indicates that this object is **shared**, i.e. via link sharing. We can list all the members that have access to that shared object by passing the id to:

`sharing_list_file_members(id)` if the object is a file, or

`sharing_list_folder_members(id)` if the object is a folder.

The returned objects lists the member users of that shared object in a list called `users`, where account details, permissions and many more information, like `time_last_seen`, are displayed.

4.2.3.3. Deleting content and restoring revisions. As already mentioned, deleting content turns the affected metadata into `DeletedMetadata`. This implies that the content is always theoretically restorable, if not deleted permanently with an business account.

It is possible to list up to 100 revisions of a file with a basic account. The command

```
revisions_list = dbx.files_list_revisions(path, limit=100).entries
```

gives a list of these revisions for a file in the Dropbox with the path `path`.

In fact, revisions are objects of the class `FileMetadata` we saw earlier. The size field of this objects is small, which leads to the conclusion that only the differential to the previous version is saved in the content.

As before mentioned, folders don't have revisions and therefore only the original version itself exists.

It is possible to restore any revision that is not older than 30 days (basic account) or than 180 days with a business account.

We can either restore a revision of a file by specifying the destination `path` on the dropbox and a revision number `rev` to restore with `files_restore(path, rev)`, but this leads to a high degree of **interference** as each revisions has to be restored on the Dropbox.

So by searching the documentation we find that we can also download a certain revision directly to our local system. In the next section we want to realize that when downloading all content from the Dropbox.

4.2.3.4. Downloading all content to local system. In this section, we focus on downloading all content of the Dropbox with as little interference as possible. We specify a root folder on the local system, in which the Dropbox content will be mirrored.

So having all metadata content pulled from an endpoint, we can now loop through all metadata objects in our list. For each metadata, we decide what to do depending on the type. So it makes sense to loop the `FolderMetadata` and create all folders first.

For each file we restore it's current state and all of its revisions that are retrievable, additionally tagging these with the `server_modified` timestamp in their filename.

The function

`dbx.files_download_to_file(local_path(m), m.path_lower, rev=m.rev)`

downloads the file with metadata `m` to any local system path we specify, but in this case within the local root folder we created before.

Special care has to be taken with files that cannot be downloaded directly, for example Dropbox Papers. These files have to be exported first, in case of a Dropbox Paper the target format might be HTML. While the Dropbox Python SDK offers the function `dbx.files_export_to_file(local_path, dropbox_path)`, this can only be used to export and download the current version of a file, it is not possible to specify a revision. In order to export and download all revisions of a file, one would have to restore each first on the server and then download it. As this procedure would interfere with our goal to induce no interference on the state of the Dropbox while cloning its content, we opted to simply inform the user about the still available revisions, and leave it to them to decide whether and when to alter the state of the Dropbox.

As mentioned before, this reduces the interference with the data on the Dropbox a lot as we don't have to use the Dropbox as intermediate storage, because we can download each file with a certain revision number directly to our local system.

4.2.4. Business Team Account Analysis. Because of the Dropbox Business account, the functionalities were expanded in the following areas:

- Access to the core features
- Advanced data protection
- Advanced sharing and collaboration tools
- Admin controls for additional security
- Support

The core features now include, for example, the possibility to send, sign and store eSignature documents. In the data protection section, it is now possible to recover deleted files or restore previous file versions which are up to 180 days old instead of 30 days. In addition, the admin can control the number of connected devices and can control what happens to disconnected ones. Dropbox paper is an additional sharing and collaboration feature to create, share, and keep your team in sync. Furthermore, blocking others from editing a file while it is being edited at the moment. Dropbox Business allows teams to have multiple admin roles, each role with a different set of permissions. In addition, a few support features will be provided, for example, the live chat to be able to ask questions quickly.

After elevating the basic account to business, we now can use the object of the class `DropboxTeam`. This object has some more functionality concerning the use of Dropbox within a team.

In the application settings there are now some more permissions we can set. As before, we give again full permissions. These embrace:

- Team: Permissions that allow your app to access basic team information
- Team Data: Permissions that allow your app to manage the team space and team member access permissions
- Files: Permissions that allow your app to view and manage your team's files and folders
- Members: Read team memberships and member settings
- Sessions: Permissions that allow your app to view and manage linked web, device, and app

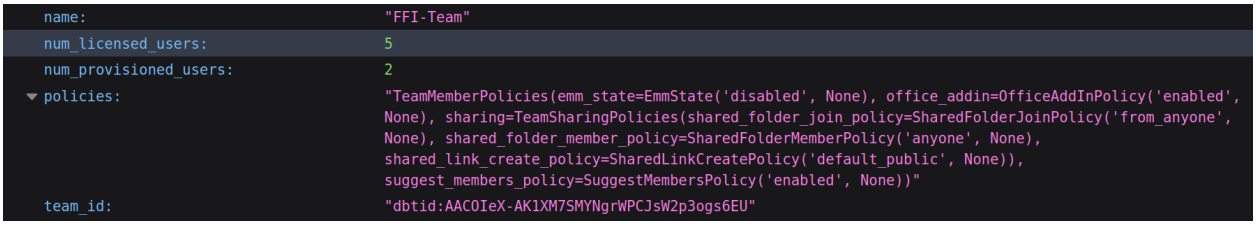


Figure 21: retrieved `TeamGetInfoResult` in JSON format

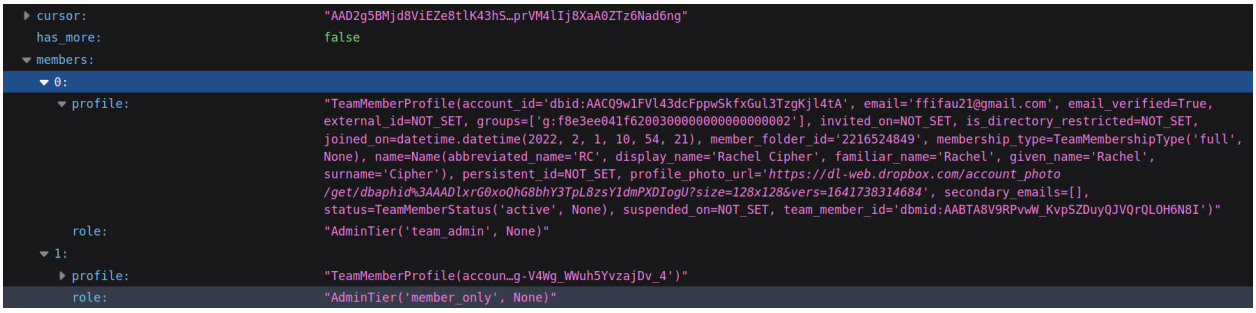


Figure 22: retrieved `MembersListResult` in JSON format

For the analysis we had to generate a new token, as there are new permissions set now and these also affect team access. So again we choose no expiration for this new token. The object we want can be instantiated with `dbx_team = dropbox.DropboxTeam(OAUTH2_TOKEN_TEAM)`.

In the documentation we look for interesting functionality regarding our goal of analysis. In the following sections we will look closer to these functions and the information we can obtain.

4.2.4.1. team info. We can get the information about the team itself with `dbx_team.team_get_info()` which returns an object of class `TeamGetInfoResult`. Figure 21 shows us the JSON exported return value of that function. We see that the fields describe the name and the id of the team, the size and even the policies that are valid within it.

4.2.4.2. team members. To get all the members of the team, even the deleted ones, we call `dbx_team.team_members_list(limit=1000, include_removed=True)` to retrieve the information embedded in the object of class `MembersListResult`. Figure 22 shows the JSON exported return value which defines the profile and the role of a member.

The first member has an admin role. The field with the profile data give some information about the account-id, the email, the joined date, the name of the user and the profile photo URL.

We see that a second member has only a member role.

4.2.4.3. team devices. We can retrieve all devices used by a team by calling `dbx_team.team_devices_list_members_devices(include_web_sessions=True, include_desktop_clients=True, include_mobile_clients=True)`.

The devices seem to be derived from the User-Agent HTTP header field used in the request. Figure 23 shows the JSON exported return value which includes some information about the logged devices and the logged web sessions.

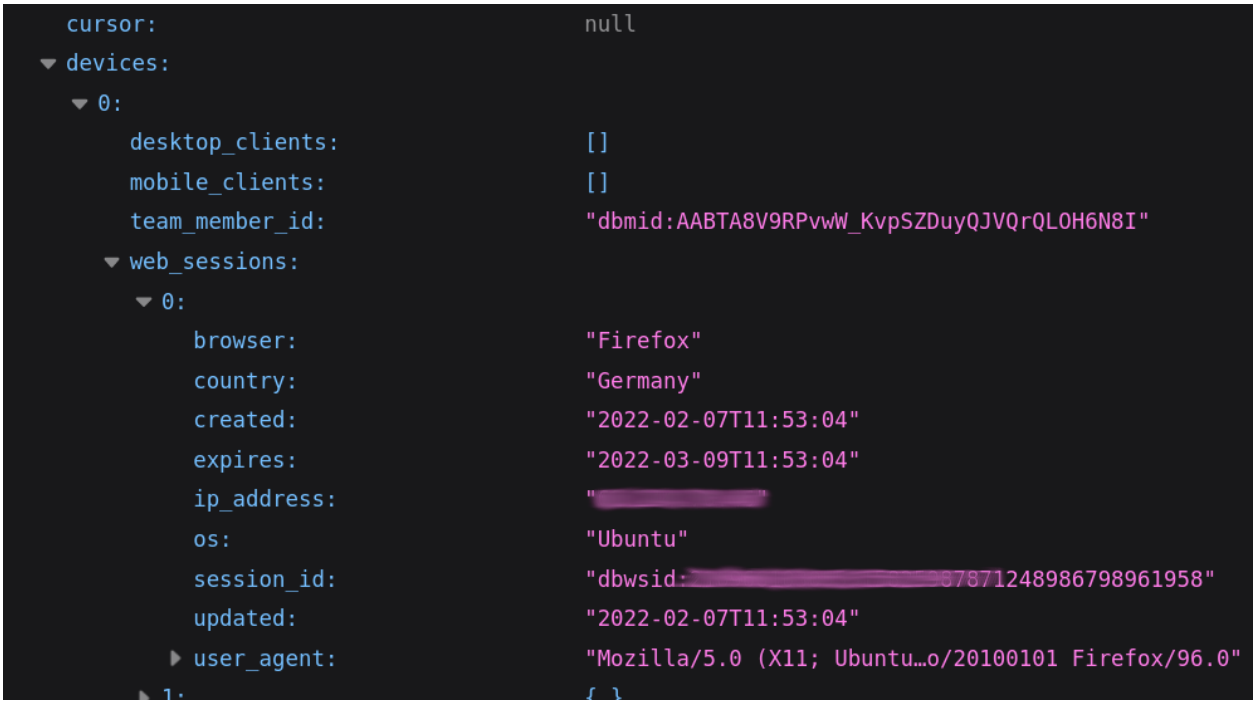


Figure 23: retrieved `ListMembersDevicesResult` in JSON format

Event Category	Event Type
logins	login_success, logout
file_operations	file_preview, file_download, file_revert, file_add, file_rename, file_edit, file_permanently_delete
apps	app_link_user
sharing	shared_link_create, shared_folder_transfer_ownership
devices	device_change_ip_web
members	member_change_status
paper	paper_doc_download
...	...

Table 3: encountered event categories and their event types

The relevant fields show the id of the team member, the used browser, the country, some timestamps, the ip-address, the operation system and the web-session-id with the user agent used.

4.2.4.4. team logs. Calling the function `dbx_team.team_log_get_events(limit=1000)`, the object `GetTeamEventsResult` that is returned holds the relevant log details in the list `events`, which is a member of this object. If this object's field `has_more` is set to true, we can use its field `cursor` to pass it to the `team_log_get_events_continue(cursor)` function, which gives the next object of class `GetTeamEventsResult` with its event information. This is repeated until the last object's field is set to: `has_more = False`.

The events are tagged with an `EventCategory` and `EventType` object. We encountered seven different event categories in our analysis, where each event type directly relates to one event category: The last line marks that there are most probably more event types that we haven't discovered in our analysis.

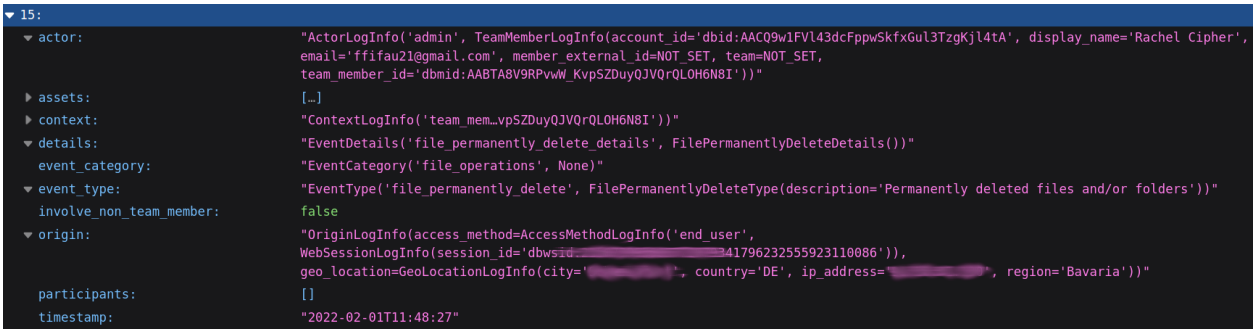


Figure 24: single entry of the list `GetTeamEventsResult.events`

Figure 24 displays a JSON exported single event of the list `GetTeamEventsResult.events`.

The event category and event type show that a file or folder was deleted permanently, where the affected path can be retrieved from the field `assets`.

The field `actor` tells us that Rachel Cipher with the mail address ffifau21@gmail.com has done this operation. Next to the given timestamp field that tells us when this operation was done, the event object also delivers us information about the geo-location, derived from the used IP Address, and about the web session.

Generally speaking, the logs provide **huge auditability**, as the timestamp, the information about the actor, the geo-location and the type of operation done can be retrieved easily together.

4.3. Summary

4.3.1. Auditability. The available field `content_hash` in the metadata structure of a file, allows our script to verify the integrity of a downloaded file. Through the revisions of a file and their recorded timestamps, it is possible to get a hold of earlier versions of a file and also track when it was changed. It is also possible to restore deleted files as long as they are not permanently deleted.

When the account is elevated and have the proper role and permissions within a team, one can list up all the events that took place. These events are accompanied by a timestamp, the actor, the estimated origin with the help of the IP address, and the affected path if the operation was done on file level. This leads to a huge increase in auditability, because every action in a team and on the Dropbox is traceable.

4.3.2. Interference. On the file level we virtually observed no interference, as no access timestamps are changed and all content can directly be downloaded to the local system without touching the Dropbox's state.

By calling the team-functions with our elevated account, we also sensed no traces of interference with the Dropbox's state, but in the team logs there is some as our application's operations are also logged, i.e. when downloading files.

4.3.3. Limitations. The availability of earlier versions of a file strongly depends on the type of account the Dropbox is linked to. For example basic accounts only hold earlier versions up to 30 days, while business accounts may hold them for up to 10 years.

It is possible to delete files permanently, which leaves no way to restore them. But on business accounts this leaves a trace in the activity log.

The activity logs are only available for business accounts, more precisely for team members with adequate permissions granted, and these logs only list the activity within a certain team.

5. Google Drive

Die Analyse von Google Drive beschreibt zunächst die vorausgehende Literaturrecherche, um Hintergründe zu erläutern.

5.1. Grundlegende Literaturrecherche

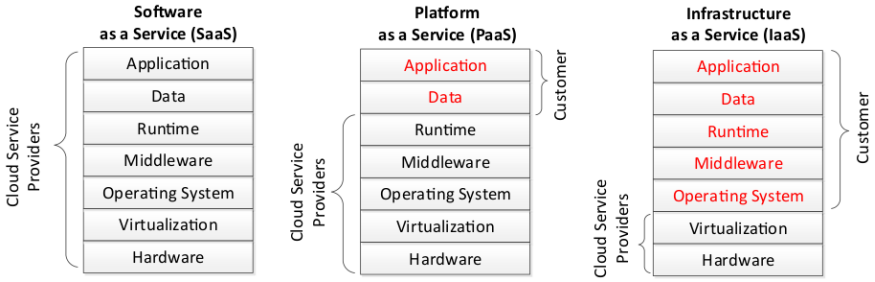


Figure 25: Cloud Computing kann in drei Modelle kategorisiert werden.

5.1.1. Funktionsanalyse von Kumodd. Kumodd ist ein Werkzeug, um Daten von Google Drive mit Blick auf forensische Anforderungen zu speichern. Unter https://kumodd.readthedocs.io/en/latest/ kann die Dokumentation von Kumodd eingesehen werden.

5.1.1.1. Dateien auflisten und herunterladen. Kumodd ermöglicht die Auflistung der verschiedenen Dateien innerhalb der Cloud. Dazu wird der Befehl **kumodd -list** < **option** > verwendet. Als Optionen kann entweder nach allem mit `all` oder nach spezifischen Dokumenttypen wie `doc, xls, ppt, text, pdf, office, image, audio, video` gesucht werden. Selbige Optionen gibt es auch für den Download der Dateien, welcher mit dem Befehl **kumodd -download** < **option** > realisiert werden kann.
Beim Download können die Dateien auch mittels der Funktionalität **-convert** < **option** > in ein anderes Format konvertiert werden. Dies betrifft spezielle Google Drive Dateiformate wie `Google Docs, Google Sheets, Google Slides und Google Drawings`. Es ist möglich diese in `pdf, opendocument, officedocument, epub, rtf, html und plain` umzuwandeln.
Es sei erwähnt, dass Kumodd nicht nach Namen der Datei herunterlädt, sondern nach Versionsnummer. Dadurch ist es möglich gleich benannte Dateien, welche ggf. sogar gelöscht sind, auf einmal herunterzuladen und zu unterscheiden. Kumodd ändert beim lokalen Erstellen der Dateien anschließend die Namen, sodass diese auch dort einfacher unterschieden werden können.
Mit der Funktionalität **-q "**< **option** > **'tag"** kann man nach Schlagbegriffen Dateien suchen und auswählen. Die dazugehörigen Optionen sind:
`title contains, not name contains, fullText contains, modifiedTime >, viewedByMeTime >` und die Option `in writers` bei welcher Tag- und Optionsreihenfolge bei Befehlseingabe vertauscht sind.

5.1.1.2. Timeline Funktionalität. Mit der Option `-l2t` kann zusätzlich eine CSV im log2timeline-Format erzeugt werden. Diese kann anschließend von weiteren forensischen Softwares zur Analyse und Auswertung von Timestamps genutzt werden. Dazu werden die Timestamps aus den GDrive-Metadaten verwendet.

5.1.1.3. Weitere Funktionen. Mit **kumodd -csv** < **csv-file-name** > können Dateien in eine vorher erstelle CSV Datei heruntergeladen werden. Mit **kumodd -verify** < **option** > können der MD5-Hash der Datei und Metadaten, sowie Größe und gewisse Timestamps verifiziert werden. Mit **kumodd -p** < **ordner** > kann der Zielordner für Download-Daten definiert werden. Mit **kumodd -m** < **ordner** > kann der Zielordner für Metadaten definiert werden.

5.1.1.4. Grenzen von Kumodd. Die Google API limitiert die Anzahl möglicher Anfragen auf 1000 pro User pro 100 Sekunden. Auch Kumood kann diese Grenze nicht überschreiten, auch wenn es unwahrscheinlich ist, dass diese Grenze erreicht wird. Für die Entwicklung eigener Software liefert dies aber die Erkenntnis, die Anzahl an Anfragen zu minimieren.

5.1.2. Kumodocs. Unter https://github.com/kumofx/kumodocs kann kumodocs genauer analysiert werden. Kumodocs ermöglicht die forensische Analyse der Logdateien zu gewissen Google-spezifischen Dokumenten. Ein Beispiel ist Google Docs, welche das webbasierte Editieren von Dokumenten ermöglicht. Dabei werden die Aktionen von Löschen, Einfügen, etc. detailliert gespeichert. Kumodocs kann diese Logs für weitere forensische Analysen nutzen.

5.1.3. Kumofs. Leider konnte auch nach erschöpfender Suche kein Quellcode zu Kumofs gefunden werden. Die Analyse basiert daher nur auf den Aussagen aus „Cloud forensics Tool development studies & future outlook“.
Kumofs bietet eine POSIX-Schnittstelle zwischen dem lokalen Gerät und der Cloud, was man sich so vorstellen kann, als würde man den Cloud Storage mounten. Dadurch können sowohl bestehende Tools verwendet werden, aber auch Kumofs selbst nutzt dies für eigene Funktionalitäten: Download, virtuelle Dateien und Ordner, Analyse gelöschter Dateien und eine Timetravel-Option. Diese Zeitreiseoption ermöglicht die Analyse und Betrachtung des Cloudspeichers zu einem vorherigen Zeitpunkt.

5.1.4. Analyse der Umsetzung von Funktionalitäten und Ideen im Bereich Cloud Forensik. Im Paper „A Systematic Survey on Cloud Forensics Challenges, Solutions, and Future Directions“ wurden die Ansätze, Tools und Ideen zur Cloud Forensik strukturiert analysiert und dargelegt. Für die eigenen Ideen sollte man also zuerst hier reinschauen, um zu überprüfen, ob es nicht bereits Literatur dazu gibt.

5.1.5. Analyse der Benutzeroberfläche „Google Cloud Platform“. Auf der Internetseite „console.cloud.google.com“ wird eine Benutzeroberfläche für die Analyse des Cloudservices geboten. Diese wurde in Anspruch genommen um mögliche weitere Daten über die Anzahl der Benutzer, die auf dem Google-Drive-Speicher zugegriffen haben, die Verwendung des Google-Drive-Speichers als auch die Aktivität der Benutzer zu sichern. In der vorhandenen Sidebar wurden zwei hilfreiche Sidebarelemente ermittelt, das Element „Logging“ und das Element „Monitoring“.

5.1.5.1. Logging. Unter Logging ist es möglich sowohl ausführliche Fehlermeldungen als auch Informationen zu erfolgreichen Aktionen zu finden. Alle Log-Einträge haben unter anderem einen `Zeitstempel`, `ID und protoPayload`, in denen man beispielsweise den genauen Namen der Methode sehen kann, die diesen Eintrag erstellen ließ, aber auch die E-Mail-Adresse, die die Methode ausgeführt hat. Auch die IP-Adresse des Aufrufers und genauere Daten über den User Agent, der für die Verbindung verantwortlich ist, wie zum Beispiel „Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:95.0) Gecko/20100101 Firefox/95.0,gzip(gfe)“.

Alle diese Log-Einträge können von der Webseite heruntergeladen werden, wenn man einen Account besitzt, der die nötigen Ansprüche erfüllt. Da wir jedoch so einen Account nicht hatten, musste die

Untersuchung in dieser Richtung aufhören.

Eine praktische Verwendung der Log-Dateien ist zudem, dass man die Spuren, die man beim Untersuchen des Cloudspeichers möglicherweise selbst hinterlässt, nachverfolgen kann, indem man eine neue Senke und einen neuen Bucket erstellt und alle neuen Log-Einträge in diesen laufen lässt.

5.1.5.2. Monitoring. Unter dem Sidebarelement Monitoring kann man sich im Metrics-Explorer entweder mithilfe der Benutzeroberfläche oder durch die „Monitoring Query Language“(kurz: MQL) verschiedene Daten veranschaulichen lassen. So kann man sich beispielsweise die Google-Drive-Anfragen zum Auflisten oder Downloaden der Daten ansehen. Zudem sind weitere Informationen wie die „credential_id“ zu finden, mit der man den „OAuth2-key“, der verwendet wurde, bestimmen kann.

5.2. Metadaten

Google Drive speichert zusätzlich zu den Dateien eine große Menge an Metadaten. Allerdings haben nicht alle Dateien die gleichen Metadaten. In diesem Kapitel wird analysiert, welche Dateien welche Metadaten haben.

5.2.1. Untersuchte Artefakte. Zur Metadatenanalyse wurden viele verschiedene Dateien in Google Drive hochgeladen. Auf diesen Daten wurden dann unterschiedliche Aktionen ausgeführt, um zu überprüfen, ob diese Aktionen über die Metadaten nachvollzogen werden können.

Zu den untersuchten Dateien gehören:

- Videodateien: avi, mov, mp4, ogg
- Sounddateien: mp3, ogg
- Bilddateien: ico, gif, jpg, png, svg, tiff, webp
- Dokumente: odt, doc, rtf, pdf, google-docs, txt, ppt, xls
- Archivdateien: zip,
- Andere Dateien: html, csv, json

Des Weiteren wurde überprüft, ob folgende Aktionen über die Metadaten nachvollziehbar sind:

- Erstellungs-, Änderungs- und Lesezeitpunkt
- Benutzer
- Gelöschte Dateien
- Geteilte Dateien

Außerdem wurden verschiedene **Versionen** von Dateien untersucht. Diese werden nur dann dauerhaft gespeichert, wenn dies explizit vom Benutzer ausgewählt wird, weswegen eine zeitnahe Speicherung aller älteren Versionen von Dateien wichtig ist.

5.2.2. Extraktion der Metadaten. Alle Metadaten werden von unserem Python-Script von Google Drive heruntergeladen und in JSON-Dateien gespeichert.

```
def _download_metadata(self, file_id, file_name, metadata_dir, include_metadata,
                                            metadata_path) -> Dict[str, Any]:
  metadata = self.service.files().get(fileId=file_id, fields="*").execute()

  if include_metadata:
    with open(metadata_path.with_suffix(".json"), "w") as f:
      json.dump(metadata, f, indent=2)
```

5.2.3. Allgemeine Analyse der Metadaten. Bei der Analyse der extrahierten Daten wurde festgestellt, dass nicht alle Daten dieselben Metadaten haben. Insgesamt wurden folgende Metadaten-Keys gefunden: capabilities, copyRequiresWriterPermission, createdTime, explicitlyTrashed, exportLinks, fileExtension, fullFileExtension, hasThumbnail, headRevisionId, iconLink, id, imageMediaMetadata, isAppAuthorized, kind, lastModifyingUser, linkShareMetadata, md5Checksum, mimeType, modifiedByMe, modifiedByMeTime, modifiedTime, name, originalFilename, ownedByMe, owners, parents, permissionIds, permissions, quotaBytesUsed, shared, size, spaces, starred, thumbnailLink, thumbnailVersion, trashed, version, videoMediaMetadata, viewedByMe, viewedByMeTime, viewersCanCopyContent, webContentLink, webViewLink, writersCanShare.

5.2.3.1. Unterschiede. der Art der Metadaten von verschiedenen Dateitypen:

Dateien	Unterschiede
Google-Dokument	Keine Metadaten für: fileExtension, fullFileExtension, headRevisionId, md5Checksum, originalFilename, webContentLink. Desweiteren ist es die einzige Datei mit Metadaten mit dem Key „exportLinks“
Bilddateien	Haben einen zusätzlichen Key für „imageMediaMetadata“
Alle getesteten Dateien außer rtf, csv, zip, mp3	Haben einen Key für „thumbnailLink“
Videodateien	Haben einen zusätzlichen Key für „videoMediaMetadata“

Interessant hierbei ist, dass Google Drive bei Bild- und Videodateien die Dateien an sich analysiert, die Metadaten extrahiert und diese in die Google-Drive-Metadaten integriert. Des Weiteren ist zu erkennen, dass Google-Dokumente (wie Google-Docs, Google-Sheets, Google-Slides, etc.) sehr unterschiedliche Metadaten gegenüber „normalen“ Dateien besitzen.

5.2.4. Einzelne wichtige Metadaten. Unter den ganzen Metadaten sind ein paar einzelne deutlich wichtiger für die Analyse des Cloudspeichers als die anderen. Daher werden im Folgenden genau diese hervorgehoben, ihre Funktion erklärt und erläutert, wieso diese wichtig für die Untersuchung sein können.

5.2.4.1. version. gibt nicht wie man vermuten kann an wie viele Versionen dieser Datei hochgeladen wurden, sondern wie viele Veränderungen an der Datei auf dem Server stattgefunden haben. Das Löschen bzw. das Verschieben der Datei in den Mülleimer zählt als Verändern der Datei. Auch das Wiederherstellen zählt dazu.

5.2.4.2. labels. ist eine Obergruppe von Metadaten. Mitglieder der Gruppe besitzen nur einen Wahrheitswert und sind:

- **modified** hat den Wert Wahr, wenn der Benutzer diese Datei verändert hat.
- **trashed** gibt an, ob diese Datei sich derzeit im Mülleimer befindet. Nur der Besitzer der Datei kann im Fall, dass der Wert wahr ist, die Datei sehen und wiederherstellen.
- **starred** sind nur Dateien, die als Favorit von dem Benutzer markiert wurden.
- **viewed** zeigt an ob der Benutzer die Datei angeschaut an.

5.2.4.3. explicitlyTrashed. ist auch nur ein Wahrheitswert, der angibt, ob die Datei im Falle, dass sie sich derzeit im Mülleimer befindet, vom Benutzer explizit dorthin bewegt wurde. False würde in dem genannten Fall meinen, dass die Datei wegen eines rekursiven Löschens dorthin gelangte, also beispielsweise, wenn man den Ordner löscht, in dem die Datei sich befindet.

5.2.4.4. lastModifyingUserName. bzw. auch **lastModifyingUser** beinhaltet den Namen des Benutzers, der die Datei zuletzt verändert hat. Dank dieser Metainformation kann man sichergehen, dass eine Person auf die Datei zugegriffen und möglicherweise teilweise oder komplett geändert hat. In Kombination mit **lastViewedByMeDate** und **modifiedDate** anderer Benutzer, die auch Zugriff auf die Datei haben, kann man auflisten, wer die aktuellste Datei gesehen hat.

5.2.4.5. owners. ist eine Liste von Benutzern mit zusätzlichen Informationen. Diese beinhalten zum Beispiel den angezeigten Namen oder die E-Mail-Adresse des Eigentümers der Datei. Bei illegalen Dateien könnte man dadurch weitere Verdächtige ausmachen.

5.2.4.6. sharingUser. ist auch eine Auflistung, die gleich aufgebaut ist, jedoch stehen in dieser Liste alle Personen, mit denen die Datei geteilt wurde. Da man ohne weitere Probleme Objekte mit Personen teilen kann ohne, dass sie davon etwas mitbekommen, bis sie auf ihren Google Drive Account schauen, kann man nicht sofort davon ausgehen, dass diese Personen Kenntnis von der Datei hatten.

5.2.4.7. createdDate. ist das Datum, an dem die Datei erstellt wurde. Wie bei einer Untersuchung eines Datenträgers, kann man durch die Erstelldaten mögliche Abläufe von Installationen und Downloads oder Unschlüssigkeit finden.

5.2.4.8. permissions. gibt die Rechte der einzelnen Benutzer an, die Zugriff auf die Dateien haben. Zu wissen, ob ein Benutzer nur Leserechte oder administrative Rechte hat, kann wichtig sein, wenn es um die Bearbeitung einer Datei oder deren Besitz sein.

5.2.4.9. originalFilename. enthält den Dateinamen, mit dem die Datei hochgeladen wurde. Der frühere Name kann Hinweise auf den ehemaligen Inhalt einer veränderten Datei geben.

5.2.5. Ähnlichkeiten zu klassischer Hard-Drive Forensik. Felder wie **createdTime**, **modifiedTime**, **name**, **permissions** und **owners** sind vergleichbar mit Metadaten, die man auch in klassischen Hard-Drive-Forensik gewinnen konnte.

5.2.5.1. Pfade. werden jedoch sehr unterschiedlich zu Dateisystemen wie beispielsweise Ext3 gehandhabt. In Ext3 gibt es für jeden Ordner eine „Directory Entry“-Struktur, welche unter anderem die IDs der Inodes der Dateien in dem Ordner enthält. In Google-Drive hingegen hat jede Datei eine Liste mit IDs von Parent-„Ordnern“. Wobei Ordner von Google-Drive als normale Dateien gehandhabt werden mit dem MIME-Type „ application /vnd.google−apps. folder“.

```
{
        "kind": "drive#file",
        "id": "1jAdZpCqzhXqF3gb4a1E7wy5jHy7ijMdz",
        "name": "folder_test",
        "mimeType": "application/vnd.google-apps.folder",
        [...]
}
```

5.3. Analyse von Thumbnails

Thumbnails werden im Webinterface dargestellt. Dazu hat jede Datei drei Metadaten:

- "hasThumbnail": boolean,
- "thumbnailLink": string,
- "thumbnailVersion": long,

5.3.1. Veränderung von Thumbnails. Kann Google Drive selbst ein Thumbnail wählen, so kann dieses nicht verändert werden. Ansonsten ist es möglich (z.b. mit einem Python Script), das Thumbnail beliebig anzupassen.

Thumbnails ändern sich nach Bearbeitung von Dateien (z.b. Word Dokumente). Mehrere Tests ergaben aber keine logisch nachvollziehbare Zeit, nach welcher die Thumbnails upgedatet werden. Es können ein paar Sekunden bis mehrere Minuten sein.

Bei der Veränderung bleibt der Link (thumbnailLink) gleich. Nur die dahinter liegenden Daten ändern sich. Die Version (thumbnailversion) zählt, wie oft das Thumbnail geändert wurde. Durch die Verzögerung der Aktualisierung der Thumbnails, kann die lokale Version different zur Online-Version sein.

Fast alle Dateien haben ein Thumbnail. Nur wenige (z.b. Zip-Datei) haben keines. In diesem Fall gibt es das Attribut thumbnailLink nicht und die thumbnailVersion ist 0.

5.3.2. Integritätserhaltung. Da sich die Daten hinter den Thumbnails ändern können, sollten für die Erhaltung der Integrität zusätzlich die aktuellen Thumbnails heruntergeladen und lokal gespeichert werden. Ansonsten kann eine Korrektheit der Thumbnails nicht gewährleistet werden.

Dies wurde bereits im Programm zum Download ergänzt und es wird im Zielordner ein weiterer Ordner erstellt, welcher beim Download zusätzlich noch die Thumbnails jeder Datei, falls es diese gibt, speichert.

5.4. Kommentare

Google Drive erlaubt seinen Nutzern Kommentare zu Dateien abzugeben. Diese werden dann an der Datei angezeigt oder im Falle von Googles eigenen Formaten wie z.B. Google Presentations auch an bestimmten Stellen. Nutzer können auch auf bereits bestehende Kommentare antworten, ihre eigenen Kommentare löschen oder als ”resolved” markieren. Kommentare, die ”resolved” sind, werden ausgeblendet.
Diese Kommentare lassen sich per Google-API in Form von json-Dateien herunterladen, wobei für jede Datei mit Kommentaren eine json-Datei angelegt wird. Beispielhafte Inhalte einer dieser json-Dateien kann man im Folgenden sehen.


```
{
    "kind": "drive#comment",
    "id": "AAAAUaEO5Sc",
    "createdTime": "2022-01-20T15:57:08.325Z",
    "modifiedTime": "2022-01-20T15:57:08.325Z",
    "author": {
      "kind": "drive#user",
      "displayName": "Forensic Crew3",
      "photoLink": "//lh3.googleusercontent.com/a/AATXAJ[...],
      "me": false
    },
    "htmlContent": "Hier ist seine Freundin",
    "content": "Hier ist seine Freundin",
    "deleted": false,
    "resolved": false,
    "anchor": "[null,[null,[0.011857707509881424,[...]"]",
    "replies": []
  },
```


Jeder Kommentar hat eine eindeutige ID und auch Zeitstempel, die Anzeigen wann der Kommentar erstellt und wann das letzte Mal bearbeitet wurde. Des Weiteren wird angegeben, wer der Autor des Kommentars war und was der Inhalt der Datei war (content/htmlcontent). Auch kann man sehen, ob der Kommentar resolved wurde, also ”unsichtbar” ist. Der Eintrag ”replies” enthält eine Liste an weiteren

Kommentaren, die auch der gezeigten Struktur folgen. Interessant ist der Eintrag ”deleted”. Denn falls ein Kommentar gelöscht wurde, lassen sich trotzdem noch Reste über die API finden, da diese gelöschten Kommentare auch noch Einträge hinterlassen:

```
{
  ”kind”: ”drive#comment”,
  ”id”: ”AAAAUzT5A2k”,
  ”createdTime”: ”2022-01-26T23:43:18.983Z”,
  ”modifiedTime”: ”2022-01-27T00:06:00.787Z”,
  ”deleted”: true,
  ”replies”: []
}
```

Wie man sehen kann, ist bei gelöschten Kommentaren zwar der Inhalt und der Autor nicht mehr ersichtlich, aber es lässt sich dennoch zeigen, dass es gelöschte Kommentare gab und auch wann diese Kommentare erstellt und gelöscht wurden (letzte mTime).
Generell lassen sich über die API Informationen extrahieren, die über die gegebenen Attribute (Zeitstempel und Autor) exakt beweisen lassen, wer wann einen Kommentar geschrieben hat. Des Weiteren kann man auch aus forensischer Sicht sehr wertvolle Informationen über Löschungen feststellen, auch wenn diese Unvollständig sind.
Im Bezug auf Interference haben sich keine Auswirkungen des Zugriffes über die API auf die erhaltenen Daten feststellen lassen.

5.5. Dateidownload

Es wurde ein Python-Skript entwickelt, um über die Google-Drive-API die Dateien mit den Metadaten zu extrahieren. Der Code zum Nutzen der Google-Drive-API, um herauszufinden welche Dateien dort gespeichert sind, ist im Folgenden abgebildet:

```
def _get_files_from_gdrive(self) -> Tuple[List[Any], Dict[str,Path]]:
  results = self.service.files().list(
    pageSize=1000, fields="nextPageToken, files(*)").execute()
  items = results.get('files', [])
  paths = self._get_paths_for_files(items)
  return items, paths
```

Der Code zum Erstellen des Requests für den Download ist im Folgenden abgebildet:

```
file_id, file_name = file["id"], file["name"]
if metadata["mimeType"] not in EXPORT_MIME_TYPES:
  # download normally
  request = self.service.files().get_media(fileId=file_id)
else:
  # export
  request = self.service.files().export_media(fileId=file_id, mimeType=
                                        EXPORT_MIME_TYPES[metadata["mimeType"]])
```

Beim Erstellen des Requests wird zwischen normalen Dateien und Dateien, die vor dem Download extrahiert werden müssen, unterschieden. Einige Dateien (Mime-Type: application /vnd.google−apps.map, application /vnd.google−apps.form) sind nicht exportierbar. Diese Dateien können nicht ohne die Google-Web-Apps verwendet werden (Google-Maps und Google Surveys).

5.5.1. Revisionen. Zudem wurde ein Python-Skript erstellt, das alle Versionen aller Dateien downloadet. Um unerwünschte Überschreibungen von Dateien zu verhindern, wurden die Dateien beim Herunterladen nicht nur mit dem Originalnamen, sondern auch mit der Versionsnummer im Namen abgespeichert.
Die Metadatenelemente von älteren Versionen sind nicht identisch zu den Metadaten der aktuellen Versionen. Sie beinhaltet deutlich weniger Elemente und somit auch deutlich weniger Informationen.

5.6. Analyse von Freigaben

GDrive bietet die Möglichkeit Dateien und Ordner mit anderen zu teilen, also diese an Dritte weiterzugeben. Dabei können illegale Inhalte verbreitet werden und Dritte würden diese dadurch auch besitzen und sich ggf. strafbar machen.

5.6.1. Freigeben von Dateien. Nun lassen sich illegale Dateien darüber verbreiten und Dritte könnten diese ohne Probleme nutzen. Eine Auflistung der Metadaten kann dabei anzeigen, welche Nutzer mit welchen Rechten Zugriff auf diese Daten haben:

```
[...]
"viewersCanCopyContent": true,
"copyRequiresWriterPermission": false,
"writersCanShare": true,
"permissions": [
{
        "kind": "drive#permission",
        "id": "08724634199296031163",
        "type": "user",
        "emailAddress": "forensic.crew.3@gmail.com",
        "role": "writer",
        "displayName": "Forensic Crew3",
        "photoLink": "https://lh3.googleusercontent.com/a/default-user=s64",
        "deleted": false,
        "pendingOwner": false
},
{
        "kind": "drive#permission",
        "id": "17153821934402067603",
        "type": "user",
        "emailAddress": "forensic.crew.2@gmail.com",
        "role": "owner",
        "displayName": "Forensic Crew",
        "photoLink": "https://lh3.googleusercontent.com/a/default-user=s64",
        "deleted": false,
        "pendingOwner": false
}
],
[...]
```

Man könnte nun fälschlicherweise die freigegebenen Nutzer strafrechtlich verfolgen. Weitere Analysen haben dies aber relativiert.

Nach einmaligen Entfernen beim freigegebenen Nutzer, hat man als durchschnittlicher Nutzer keine Möglichkeit mehr, auf diese Dateien zuzugreifen. Trotzdem bleibt in den Metadaten die „permission“ nach dem Entfernen erhalten. Der Herausgeber könnte also im Nachhinein die Datei verändern, sodass diese illegal ist und den Verdacht auf freigegebene Nutzer lenken, obwohl diese bereits nicht mehr darauf

zugreifen können. Somit ist dies kein ausschlaggebendes Indiz und eine Log-Analyse des Nutzverhaltens ist zusätzlich notwendig.

Selbiges gilt für kurzfristige Verbreitung von illegalen Daten, weshalb auch immer auf die jeweiligen Zeitstempel, bzw. Nutzungslogs geachtet werden muss. Stelle man sich vor, jemand weiß von der Untersuchung eines Drittens und übermittelt diesem kurzfristig illegale Daten. Dann hatte dieser keine Möglichkeit auf die genannten Daten überhaupt zuzugreifen oder dies wahrzunehmen.

Bei Löschung der Dateien bei freigegebenen Nutzern gibt es keine Veränderungen bei anderen Nutzern.

5.6.2. Freigeben von Ordnern. Für die Löschung des kompletten Ordners gelten die selben Faktoren wie bereits für Dateien.

Relevant ist aber die Veränderung von Dateien innerhalb einer freigegebenen Ordnerstruktur. Alle Nutzer mit Schreibrechten können dabei Veränderungen an den Dateien und damit auch Löschungen vornehmen. Diese Löschungen innerhalb des Ordners übertragen sich auf alle Nutzer des Ordners. Eine Datei wird somit nutzerübergreifend gelöscht. Diese könnte dazu führen, dass Dritte strafrechtlich relevante Dateien während der Untersuchung löschen und diese Beweise dabei vernichtet werden.

Unklar bleibt dabei, in welcher Form Spuren in den Log-Dateien hinterlassen werden. Relevant ist die Fragestellung, ob nur beim ausführenden Nutzer Spuren entstehen oder sich diese auf alle Teilnehmer übertragen.

6. OneDrive

6.1. Introduction

Microsoft OneDrive (formerly SkyDrive) is a Cloud Storage Service that allows access from anywhere using a web-browser or its dedicated App, which is available for Windows, Windows Server, MacOS as well as Android and iOS [21]. It is a file hosting and synchronization service, that permits sharing. It is somewhat connected with other Microsoft products that are part of Office365, which enables lots of and easy interplay between those applications. [1] Every Windows 10 operation system is issued with a client of OneDrive, integrated in the Windows Explorer, ready to use. Every user has a Basic and free space of 5GB to use. Services like OneDrive can be abused by criminals, for e.g. the distribution of copyright materials [6].

Microsoft offers their service in two different variants: OneDrive – Personal and OneDrive for Business. As the name implies, the former is tailored towards private customers, the latter towards businesses and companies.

Security is ensured using state-of-the-art techniques. These include access control (i.e. engineers aren't granted permanent access) as well as security monitoring systems (e.g. real time monitoring, records of requests, internal penetration tests). However, some features, such as two-factor authentication, are optional and not enforced. Furthermore, data is protected in transit (using TLS[5] encryption and enforcing HTTPS) and at rest (physical, network & firewall protection, application security, content protection). At rest, the data is encrypted using AES256 encryption, whereas the master key is stored in Azure Key Vault. While one remains owner of their data, this is not a so-called "zero-knowledge" encryption, which means Microsoft could technically decrypt and read all stored data.

Another important aspect of OneDrive's feature is the OneDrive Personal Vault. This is a special protected area within OneDrive that enforces additional security measures (e.g. two-factor authentication). Additionally, all data is encrypted using Bitlocker, in case it is stored on a local hard drive.[20]

5. Transport Layer Security

6.1.1. Microsoft Graph REST API. [19] Microsoft's Graph REST API provides access to data of most Microsoft cloud services, including OneDrive, and offers a single endpoint (https://graph.microsoft.com). Using its functionality, the proof-of-concept tool (see chapter 6.2) has been developed. Therefore, the most important API features are the following:

6.1.1.1. Drive. Represents a logical container of files (e.g. document library, a user's OneDrive); top-level object within a user's OneDrive.

6.1.1.2. DriveItem. Represents an item within a drive (e.g. document, photo, video or folder); objects inside a drive's file system.
DriveItems can be accessed via their unique identifier or a file system path.

6.1.1.3. Data exposure. Both *Drive* and *DriveItem* can expose data in different ways:

- **Properties** expose simple values (e.g. *id*, *name*)
- **Facets** expose complex values (e.g. *file*, *photo*)
- **References** point to collections of other resources (e.g. *children*, *thumbnails*)

6.1.1.4. Delta. provides functionality to track and list changes in a *drive* or *driveItem* as well as its children.

6.1.1.5. JSON Representations. of the different resources are attached in Appendix A. An overview is presented in Figure 26.

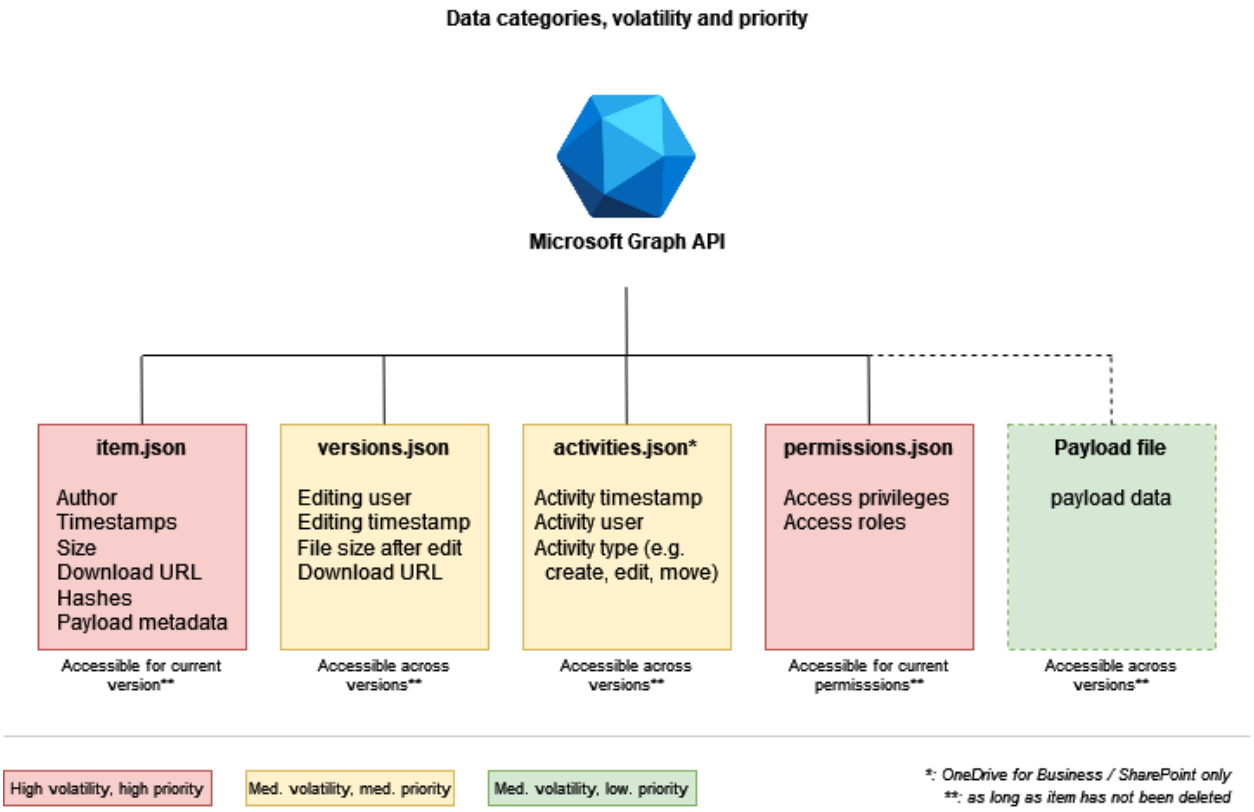


Figure 26: Data Categories

6.2. Tool

In order to evaluate strategies of taking forensic snapshots of a given OneDrive account, a proof-of-concept (PoC) tool has been developed. As the tool is implemented as a PoC, it aims to demonstrate some possibilities of Microsoft OneDrive forensics. Therefore, the tool does not guarantee to be stable on every platform, with every input and/or every cloud drive size. Nevertheless, it is assumed that all features can be implemented in such way, that the resulting tool is stable and input-independent.

Having a conceptual view on the tool, it mainly implements three steps to produce a forensic snapshot: Authentication, snapshot, post-processing.

6.2.1. Authentication. For accessing the Microsoft Graph API, it is necessary to pass a valid access token. This access token can be retrieved by the Microsoft Single-SignOn (SSO) service located at `login.microsoftonline.com`. When accessing the SSO service via HTTPS, it is possible to provide a GET parameter containing another redirection URL. After the authentication, the redirection URL will be requested with an attached GET parameter that specifies an *authentication code*. Using this *authentication code*, it is possible to receive the access token as well as the refresh token. As the access token expires after a specified period of time, the refresh token can be used for a simplified renewal of the access token. Once the refresh token has been retrieved, the renewal can be processed in the background without any user interaction.

From a forensic point of view, a helpful feature has been implemented by Microsoft: During the authentication procedure, it is possible to specify the scope of the requested authentication code. Hereby, it is possible to request read only access that prevents the snapshot tool to make unintended modification to the cloud storage.

After a successful authentication, all cloud drives of the user can be accessed using Microsoft Graph API.

6.2.2. Snapshot. Multiple concepts have been implemented in this step of the investigation process. In order to decrease the required time for the forensic snapshot, multiple concepts have been implemented in this step of the investigation process. Each API or download request that will be executed, will be logged including the timestamp, the request method, the requested URL, the responded HTTP status and the responded HTTP body size.

6.2.2.1. Sequential snapshot. As the most primitive strategy, an iteration over all drives and drive items will be considered. During this iteration, the meta data as well as the actual payload data will be downloaded. As this strategy implements a step-by-step traversal of the users drive, it will be called *sequential snapshot*. This procedure has been implemented in the file `snapshot.py`.

6.2.2.2. Delta monitoring. Having only the sequential snapshot, interference issues might reduce the quality of the investigation. This problem could occur if items get modified after the snapshot has started, but before payload and metadata of such items have been downloaded.

As described, the Microsoft Graph API (compare 6.1.1) offers the possibility to monitor changes using a so-called *delta-token*. Before starting the overall snapshot procedure, the latest token gets queried by the delta endpoint of the Graph API. As this token is representing the state of the monitored drive at the time of the query, all changes that are made after the query can be reproduced.

During the snapshot process, the PoC tool makes use of a polling approach. In a predefined period of time (i.e. 1 second), the latest changes since the last polling request are queried and stored. As a result, almost each intermediate version of a drive item will be stored if the drive item gets modified during the snapshot process. It might happen that modifications are so fast that a polling interval is too slow

to track them. In this case, some intermediate versions might get lost. But it is assured that the latest version of the modified file will be snapshotted.

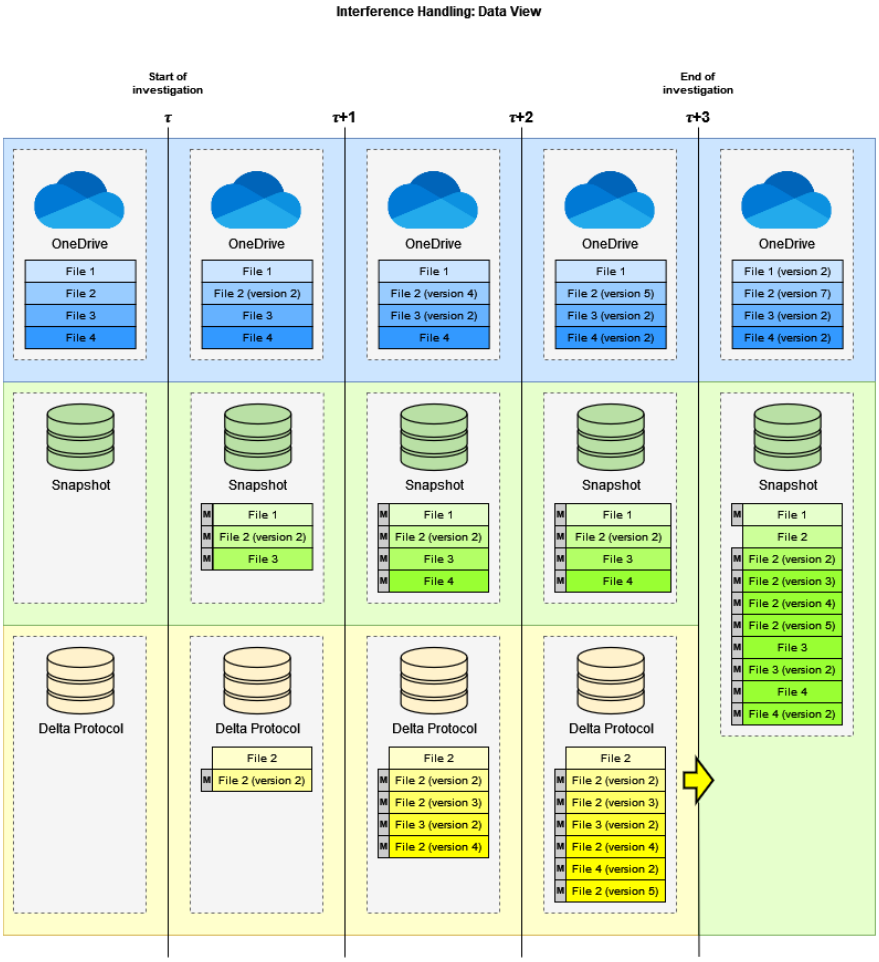


Figure 27: Interference handling using delta monitoring: Data view

Using the sequential screenshot as well as the delta monitoring in parallel, it is still not possible to produce an atomic forensic snapshot at the time when the investigations starts. As a matter of fact, it is possible to produce a pseudo-atomic snapshot when the investigations ends, since the latest version of each drive item will be stored at that time.

From a data view (see Figure 27), all drive items will be stored sequentially by processing the sequential snapshot. In parallel, all changes of the files – no matter if they have already been traversed by the sequential snapshot – will be stored separately. In the end, both snapshots will be merged resulting in a pseudo-atomic snapshot of the drive. From a processual view (see Figure 28), the pseudo-atomic snapshot has been secured at time $\tau + n$, while τ is representing the beginning of the investigation and n is representing the time that is required for the sequential snapshot.

The whole process has been implemented in the file `delta_tracker.py`.

6.2.2.3. Download prioritization. One challenge of the forensic investigation of OneDrive cloud drives is that data (payload as well as meta data) is only available as long as an item has not been deleted. Assuming cooperative infrastructure users, this is not a problem. But if a malicious user deletes drive

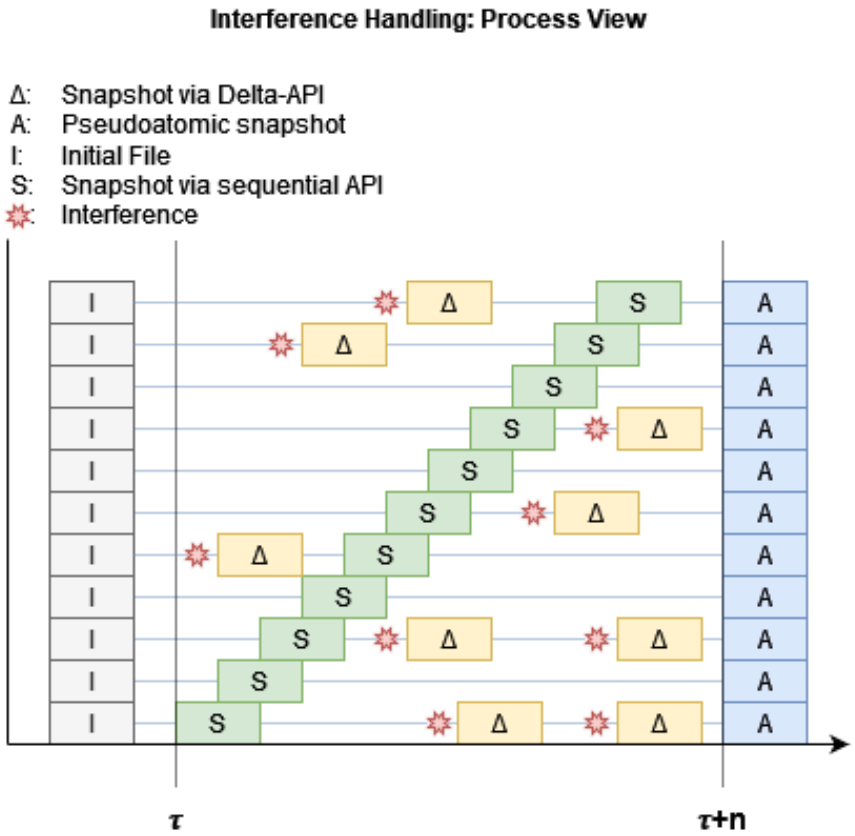


Figure 28: Interference handling using delta monitoring: Process view

items on the drive during the snapshot process, it might be possible this data will not occur in the resulting snapshot. In consequence, the quality of the snapshot increases with a decrease of the required time.

When downloading files without any order or prioritization, it might happen that the download of a 1 GB payload file will block resources in such way that 1,000 meta data files of 1 MB cannot be downloaded in that time – even if the investigative influence of one big file could be much lower than the influence of 1,000 meta data files.

Download prioritization therefore aims to download more volatile data before less volatile data (see Figure 26). Hence, a multi-threaded downloader has been implemented in the file `downloader.py`. This downloader provides three queues of different priorities. As long as a higher priority queue contains items, the lower priority queue will not be touched. Hence, the higher prioritized meta data will be downloaded before a big payload file.

In the used implementation, the *high volatility, high priority* will be downloaded before the *med. volatility, med. priority* files. All *med. volatility, low priority* files will be downloaded if none of the higher classified files are still pipelined for a download. Within these categories, all files will be treated equal.

6.2.2.4. File buffering. Considering modern computer systems, the I/O on the hard drive seems to be the bottleneck when downloading and storing much data in a short time (SSD drives offer write speeds between 150 MB/s and 600 MB/s depending on the interface and cooling, while fiber based internet allows download speeds higher than 1 GB/s). As stated before, data that has been pipelined cannot be downloaded anymore if the drive item has been deleted in the mean time. In consequence, waiting for

hard drive I/O when downloading files could delay the overall download. Hence, less information would be downloaded if a malicious user deletes drive items during this process.

Therefore, all downloaded data will be buffered in the device RAM as long as not all data has been downloaded. After the downloading procedure has been completed, the buffered data will be written to the disk. This feature has been implemented in the file `file_write_buffer.py`.

So far, the PoC tool is assuming that the investigation device has enough RAM for buffering all data, which is not necessarily valid for every combination of cloud drive and investigation device.

While writing the downloaded files to the hard disk, the SHA256 hash will be calculated and stored. In the end, all checksums will be stored in a separate file to enhance data consistency.

6.2.3. Post-processing. After all files have been downloaded, there are two main directories containing the sequential snapshot and the monitored deltas. In order to enhance the usability for investigators, further file structures will be created.

As a first step, the drive items will be flattened meaning another directory will be created that only contains the latest version of each item. In result, this directory is representing the pseudo-atomic snapshot at time $\tau + n$.

During the snapshot process, all files have been stored in a flat hierarchy of drive ID and item ID, but not using the actual path of the user's file system. Hence, the second step of the post-processing is building the file system tree using the actual payload files and the downloaded meta data. This tree will be stored in another directory and will not contain any meta data. In consequence, this directory is representing the regular user view on the cloud drive.

6.3. Experiments

At first a test-cloud-account on OneDrive has been created. As stated below, in the several experiments, specific files, compare section B table 8, were uploaded via the browser to OneDrive. Then, these files were downloaded with the created tool.

6.3.1. Interference. Some test files were created on a local computer with their respective sha256sums. These files have different formats, such as `jpg` or `txt`. After uploading and doing a snapshot, the sha256sums are compared.

6.3.2. Metadata. Two versions of the `Summieren.xlsx` file were uploaded. The file `IMG_20200217_160600.jpg` was shared by a Link. After the snapshot, a look on the possible collected metadata is made. Then the link was removed, to see what changes happened.

6.3.3. Auditability. In order to test for the auditability, we log our http requests in a `csv` file to prove, which requests we made, and how big the answer is we get.

6.3.4. Concurrency and Delta-snapshots. For testing the Delta-snapshots, a 512MB file was uploaded to the cloud-storage. While the snapshot is made, the file `test.txt` was changed multiple times to several different versions.

6.4. Results

In the following, the results of the previously explained experiments (see chapter 6.3) are listed. Details are attached in Appendix C.

6.4.1. Interference. The comparison of the sha256sum revealed identical hash sums.

6.4.2. Metadata. The MACB-times depend on the upload, i.e. when uploading a file, it is created in the cloud. Hence, these MACB-times don't correspond to the ones of the original files. However, there are no changes when downloading the files.

6.4.3. Link Sharing. If a file is (publicy) shared, this information is available in the `permissions.json` file of the respective file. If this sharing is cancelled, however, there's no way to detect a previous sharing from the available data, unless the file was changed. In that case, the `permission.json` is available for the corresponding version.

6.4.4. Auditability. All HTTP-requests are logged. They are stored in the file `request_log.csv`.

6.4.5. Concurrency and Delta-snapshots. All different versions of the `test.txt` file are accessible via the delta-snapshots.

6.5. Summary

The comparison of the sha256sums of the files yields identical checksums. Additionally, the tool has *read only* access, thus preventing any changes. Therefore, interference is avoided.

Via the Graph API, a lot more metadata is available compared to traditional storage. This includes versioning, that allows tracing of what user modified which file at which time. Unfortunately, only the user's id is logged and no device specific information, which might be of forensic interest.

However, one major drawback is, that if a file is deleted, it is gone indefinitely, including all metadata. Hence, deleting files removes all traces. Likewise, this applies to link sharing. After the shared link has been revoked, it is removed from the metadata. If the file versioning is active and the relevant versions can be restored, it is still possible to detect external modifications of the files using the versioning metadata. But if the versioning is disabled or some versions are not accessible anymore, it might be possible that external modifications can neither be proved nor disproved using the snapshot.

By logging all HTTP-requests we thrive to achieve full auditability. Assuming that the Graph API meets Microsoft's public documentation, it should be possible to reproduce all actions using only the request log and the documentation. This, nevertheless, requires some trust into Microsoft's Graph API. This also includes any errors within the API, which are not accounted for.

Since the delta functionality provides insights into any modifications to files (i.e. who and when saved which file), it is possible to attain a "pseudo-atomic" snapshot at time $\tau + n$, assuming tau is the beginning of the snapshot and n is the duration of the snapshot. While versioning is per-default active, this can be deactivated in the settings, thus hindering these outcomes. Considering the "cooperative lab environment", this may be of less concern. While it is also possible to restore intermediate versions that have been created before the investigation itself, Microsoft does not guarantee the number of stored versions or how long they will be accessible. From a forensic point of view, it is worth to mention that the delta/versioning feature only provides the information who saved the file. This user is not necessary the same person who edited the file, e.g. if two users work on the same document simultaneously and the synchronization is not based on saving the file. Assuming a real investigation, the integrity of the snapshot will not be violated if another person modifies files in the cloud storage of the accused person in order to fake evidences.

While deleting files might have a big impact on the quality of the snapshot, there is one approach to track file deletions during the investigation process: In the metadata of the drive, there is some information

about the number of bytes used as well as the number of bytes deleted. This information is assumed to have an unlimited lifetime.

As the tool aims to download metadata first, it might be possible to incriminate a suspect even if files will be deleted during the snapshot to prevent a download of compromising material. Using checksums of the metadata of each file, it is possible to drive a comparison with the checksums of known illegal files. In consequence, someone could be accused of the possession of specific media files even if these media files have not been downloaded. Furthermore, the file metadata might contain payload metadata, such as image file metadata.

Finally, while this analysis solely focused on OneDrive, these insights might be (at least partially) applicable for other services that utilize the Graph API, such as Sharepoint, Microsoft Teams and other Microsoft 365 applications.

6.6. Future Work on OneDrive

While fundamental challenges such as interference and auditability have been tackled by this report, there are still lots of forensic investigations as well as features of the Graph API that can be evaluated in the future.

This includes for instance the so-called *Webhooks*, which indicate changes within the cloud storage. Even though the modification itself isn't directly part of the webhook, it encourages to investigate e.g. the Deltas again. Thus, it is especially useful when content is altered at unknown times, or for automation purposes.

An interesting topic to explore is the behaviour of files that are removed, while a client is trying to download them. And particularly in that regard, if downloading a file with limited bandwidth could lead to delaying the deletion.

Another potential experiment that might give more insights into auditability could utilize e.g. *Wireshark* to monitor all network traffic.

Furthermore, *OneDrive Personal Vault*, a special protected area within the cloud storage (see Section 6.1), offers an additional opportunity for forensic investigations.

7. iCloud

7.1. Introduction

After we agreed on analysing all possibilities to retrieve data related information, we came up with the strategy of collecting information by

- looking for public information on iCloud drive endpoints
- using and analysing 3rd party libraries which connect to iCloud drive
- intercepting the network traffic of the iCloud drive web UI
- Implement a Python script with the found insights, to use and investigate the found endpoints in terms of interference and auditability.

7.2. Findings

7.2.1. Endpoints. Since Apple does not provide any API documentation about their endpoints, we were forced to analyze the network traffic of the iCloud web interface as well as the source code of pyicloud[6]. Below is an overview of all APIs we found and which we considered useful for a forensic file analysis.

7.2.1.1. POST /retrieveItemDetailsInFolders.

Request URL: https://p57-drivews.icloud.com/retrieveItemDetailsInFolders

Description

Retrieves meta data as well as the contents of a folder in iCloud Drive for a given folder identifier. Detailed information about this endpoint with example requests and responses can be found in Appendix D.1.

7.2.1.2. POST /retrieveTrashDetails.

Request URL: https://p57-drivews.icloud.com/retrieveTrashDetails

Description

Retrieves information about the iCloud trash folder. With the parameters and the request body that were observed in the iCloud web interface, the response seems to contain only the number of elements in the trash folder. Detailed information about this endpoint with example requests and responses can be found in Appendix D.2.

7.2.1.3. POST /storageUsageInfo.

Request URL: https://setup.icloud.com/setup/ws/1/storageUsageInfo

Description

Retrieves information about the storage usage in iCloud Drive. In particular provides information about the distribution of the used storage space with respect to different data categories (photos, backups, generic documents, e-mails). Detailed information about this endpoint with example requests and responses can be found in Appendix D.3.

7.2.1.4. POST /download/batch.

Request URL: https://p57-docws.icloud.com/ws/com.apple.CloudDocs/download/batch

Description

Retrieves file information for a given array of document_id's. The returning information consists of several attributes which are used by the iCloud web interface to directly open a file in the browser. Examples are an unique ownerId as well as a url to the specific file.
Detailed information about this endpoint with example requests and responses can be found in Appendix D.4.

7.3. Forensic Investigations Tool

7.3.1. Overview. In order to be able to examine iCloud's properties regarding interference and auditability, we introduce a command line based Python tool that allows to access the contents of iCloud Drive for a given iCloud account[7]. It provides a possibility to view and download files as well as whole folders. For every downloaded item, a SHA256 hash sum is computed and stored on the machine where the file is stored to. Furthermore, for every item in the iCloud file system, the available meta data is

6. https://github.com/picklepete/pyicloud
7. https://gitlab.cs.fau.de/be51xopo/cloudForensics-icloud-analysis

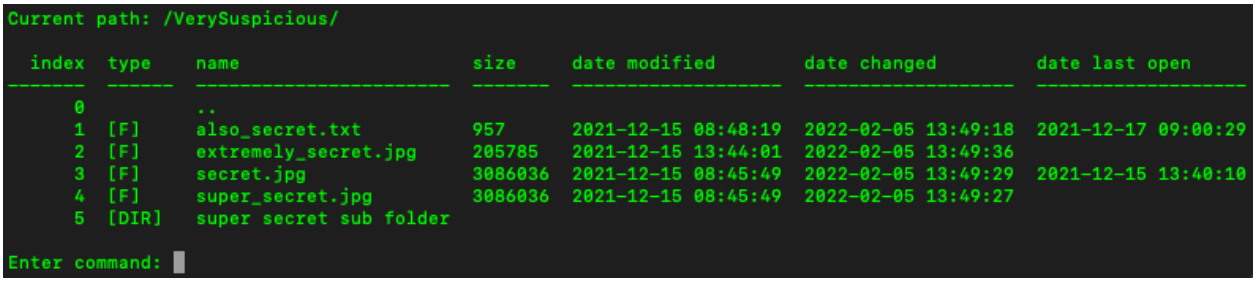


Figure 29: The tool's user interface

displayed. Additionally, the tool provides a timelining function that can be applied to selected folders or to the complete iCloud Drive contents, retrieving all the contained files' timestamps and ordering them chronologically.
In the following, an overview over the usage of the tool is given as well as some technical information.

7.3.2. Usage. When starting the tool, the user is asked to enter the desired account's credentials. If the same account is regularly accessed, it might be useful to create a file called *credentials* in the tool's root folder and write the username in the first line and the password in the second line. The tool will automatically detect the presence of the *credentials* file and use the respective credentials. After logging in successfully, the program might ask for the two factor authentication code that is usually sent to all devices connected to the iCloud account. If the code is entered correctly, access to the iCloud contents is granted and the user can start to use the tool.
The tool can be used like a REPL, i.e., it continuously waits for specific commands that are then evaluated according to the program's current state. Figure 29 shows the tool's basic user interface consisting of the currently path, the list of files contained in the current directory and the command line.

The values contained in the *index* column are not part of the files' or folders' meta data. Instead they are used as item identifiers by the tool with which the items can be referenced to in commands. Table 4 shows a list of the available commands as well as their functions.

Figure 30 shows the result of executing the *timeline* command. As can be seen, for every timestamp that is associated to one of the files contained in the directory for which the timeline is supposed to be created, the corresponding events are listed. In particular, a single file might occur up to three times in a timeline file (as each file has three timestamps). Timestamps always refer to UTC.

7.3.3. Technical Details. The tool was implemented using Python 3.7. The communication with iCloud is handled via the library *pyiCloud*[8]. *PyiCloud* accesses iCloud via an HTTP REST API as described in Section 7.2.1. The authentication is done via the endpoint https://idmsa.apple.com/appleauth/auth. Afterwards, the setup endpoint (https://setup.icloud.com/setup/ws/1) is called, which returns the URLs of the various iCloud related services provided by the API (like the iCloud Drive API). It has to be noted that we were unable to find any official or unofficial documentation of this API. The creators of *pyiCloud* do not provide information on how they obtained the API endpoints, but there are clues that at least for some parts of the library, the endpoints were obtained via the same approach that we described in section 7.2.1, i.e., by analyzing the network traffic while using the iCloud web interface[9].

8. available here: https://github.com/picklepete/pyicloud
9. see https://github.com/picklepete/pyicloud/issues/176

Command	Example	Description
any index	42	If the item with index 42 is a folder, the current directory is changed to the respective folder. If it is a file, the tool tries to open the file using a suitable external software.
download	download 42	Downloads the item with the specified index. If the item is a folder, the folder is downloaded recursively. In both cases, for all downloaded files a SHA256 hash sum is generated and stored in a file called *hashes* (by default - if such a file already exists, the user is asked to choose a file name) in the program's root folder.
download	download all	Applies the download command (see above) to the iCloud Drive root directory.
download	download .	Applies the download command (see above) to the current directory.
d	d 42	Short version of the download command (see above)
timeline	timeline foo 42	Recursively collects all files in the directory with the specified index (if the index refers to a file, an error is thrown) and sorts their three different timestamps (modified, changed, last open) ascending. The result is a file where for each timestamp the corresponding events are listed (see Figure 30 for an example output). Additionally, a SHA256 hash sum is generated and stored analogous to the *download* command.
timeline	timeline foo	Applies the timeline command (see above) to the iCloud Drive root directory.
exit	exit	Exits the program.

Table 4: List of commands that can be entered in the REPL

7.4. Interference

As mentioned above, we are investigating the degree of change to a user's data due to a programmatic forensic backup of those files with the tool presented in Section 7.3.

Apart from manually checking the results of interacting with the iCloud via our tool, we implemented a tiny Python script that prints the meta data of a specific file in a users iCloud drive, downloads the file and prints the meta data again, all using the same mechanisms as our forensic backup tool.

Figure 31 shows that there is no change to the meta data of the downloaded file at all.
Important for a forensic investigation is the fact, that the downloaded file, i.e., the file that resides on the machine of the forensic investigator, has changed meta data as it is a new file (see Figure 32). So for a forensic investigation, only the **meta data which comes directly from the cloud** should be respected.

To verify the integrity of the iCloud files even after interaction with our tool, we created SHA256 hash sums of the files and performed various actions with our tool (downloading files, creating timelines, previewing files). Afterwards we created the files' SHA256 hash sums again and compared them to the previously created ones. The hash sums of all files were identical which suggests that our tool does not alter the files' contents in any way via any interaction.

7.5. Auditability

The last part of this investigation covers the auditability of the files in a user's iCloud drive. The goal is to get an overview of the possibilities a forensics expert has, to trace and prove access and usage of the files in that cloud file system both by the iCloud account's owner as well as the investigator himself.

```
2021-12-15 08:45:49:
/VerySuspicious/secret.jpg modified
/VerySuspicious/super_secret.jpg modified

2021-12-15 08:48:19:
/VerySuspicious/also_secret.txt modified

2021-12-15 13:40:10:
/VerySuspicious/secret.jpg last opened

2021-12-15 13:44:01:
/VerySuspicious/extremely_secret.jpg modified
/VerySuspicious/super secret sub folder/extremely_secret.jpg modified

2021-12-17 09:00:29:
/VerySuspicious/also_secret.txt last opened

2022-02-05 13:49:18:
/VerySuspicious/also_secret.txt changed

2022-02-05 13:49:27:
/VerySuspicious/super_secret.jpg changed

2022-02-05 13:49:29:
/VerySuspicious/secret.jpg changed

2022-02-05 13:49:36:
/VerySuspicious/extremely_secret.jpg changed

2022-02-05 13:49:52:
/VerySuspicious/super secret sub folder/extremely_secret.jpg changed
```

Figure 30: Example result of the timeline command

```
meta data of the file in the cloud BEFORE downloading
name: % test.jpg
before download modified timestamp: % 2013-06-10 21:48:18
changed timestamp: % 2021-12-11 13:57:43
opened timestamp: % None
size % 3510397
type % file
meta data of the file in the cloud AFTER downloading
name: % test.jpg
modified timestamp: % 2013-06-10 21:48:18
changed timestamp: % 2021-12-11 13:57:43
opened timestamp: % None
size % 3510397
type % file
```

Figure 31: Script execution to show meta data changes due to forensic backup

7.5.1. Tracing actions performed by the investigator. In order to reliably trace actions performed by a forensic investigator which could be used to prove the investigator's approach, server side proofs are needed. Logging the investigator's actions on the client side is not a suitable approach to produce credible proofs as any data on the client side may be forged trivially.
To reliably trace an investigator's actions, a possibility to access server side logs would be ideal. However, we were unable to find documentation on the existence of such logs and accordingly ways to access such logs.
We furthermore tried to verify the integrity of files downloaded via our tool by creating several files on our local machine, computing a SHA256 hash sum for each of them and uploading them to the iCloud. Afterwards we downloaded the files via our tool and computed the SHA256 hash sums for the

Figure 32: Different meta data on the local file

downloaded files. The hash sums of all downloaded files were identical to those of the original files.

7.5.2. Tracing actions performed by the iCloud account owner. In order to trace actions performed in the iCloud itself, we experimentally accessed the iCloud's contents using different variations of operating systems, clients and access methods. We focused on the behavior of the three different timestamps iCloud provides, observing under which conditions the timestamps are set or updated. Table 5 shows the results of these experiments.

Step description	**date modified**	**date changed**	**last opened**
Windows (iCloud client locally installed)			
Upload file	✓	✓	✓
Create copy			
View with Windows standard viewer			
Edit with Windows standard editor	✓	✓	
Rename file		✓	✓
macOS (native iCloud support)			
Upload file	✓	✓	✓
Create copy			
View with macOS Preview via space bar			
View with macOS standard viewer (Preview / TextEdit)		✓	✓
Rename file		✓	
Add comment		✓	
Add tag		✓	
Edit with macOS Preview via space bar	✓	✓	
Edit with macOS standard viewer (Preview / TextEdit)	✓	✓	✓
Web UI iCloud			
Upload file	✓	✓	✓
Create copy			

Table 5: Steps to test auditability

8. Discussion

Network storage for files is an established technology that is used in many circumstances today. The analyses of several of such technologies above have shown, that each instance of a file storage service has its individual differences. In this section we attempt to summarize these findings and develop a classification that extends also to other network file storage services.

We begin with a summary of the different types of metadata collected/provided by the different storage technologies. We then discuss the different findings regarding the two main aspects Interference and Auditability and develop a classification scheme based on these insights.

8.1. Amount and types of metadata

In local file systems, the definition of Carrier [4] distinguishes "essential" and "nonessential" data. Essential data are "those that are needed to save and retrieve files", nonessential data are "those that are there for convenience but not needed for the basic functionality" [4]. For analysis of cloud-based or network storage, this definition must be adapted to account for the abstraction layer between a client and the raw bytes.

We are only concerned with API-readable metadata, only data that a service provider actually provides to us. This does explicitly not include information on the remote system, type of software used for file storage or infrastructure to provide files. We also focus on services that are file-based, meaning that they provide means to write, read, push or retrieve individual files with filenames and mappings to local files. This excludes rich cloud-based editors like Google Docs which do not directly handle and provide files, but abstract documents that may be exported to files.

This abstraction layer, an API controlled by the service provider, leads to a situation where no metadata at all should be considered essential data, as we do not know which data is provided as-is and which is inferred from a database or calculated on-demand. For capturing the value of the distinction between essential and nonessential data, we propose the distinction into server-controlled and client-controlled data.

If the storage service provider is to be trusted, the server-controlled data may be trusted too. Server-controlled data includes data that is written, logged or inferred by the server, without being explicitly writeable by clients. Client-controlled data is taken as-is and may be chosen freely. The actual content data, like files or pictures, is always client-controlled data in this context. We consider the following list as examples for server-controlled and client-controlled metadata. Note that not all service providers include all of the following types of data.

- Server-controlled:
  - Sharing data
  - Content hashes
  - Modification logs
  - Access logs
  - External thumbnails
- Client-controlled:
  - Optional fields like notes, comments
  - Any metadata fully contained inside files
- Data that may exist on both sides:
  - Creation time
  - Last change time
  - Access time

Type	Samba	Nextcloud	Dropbox	Google Drive	OneDrive	iCloud
Creation time	✱	✱	✗	✓	✓	✗
Last Change time	✱	✱	✓	✓	✓	✓
Access time	✱	✱	✗	✓	✓	✓
Sharing data	✓	✓	✿	✓	✿	✓
Content hashes	✗	✗	✓	✗	✓	✗
Modification logs	✗	✗	✿	✿	✿	✗
Access logs	✗	✗	✿	✿	✿	✗
External thumbnails	✗	✓	✓	✓	✓	✓

Table 6: Availability of server-controlled metadata. ✓: available; ✗: unavailable; ✱: available, but client-controlled; ✿: only available on business accounts;

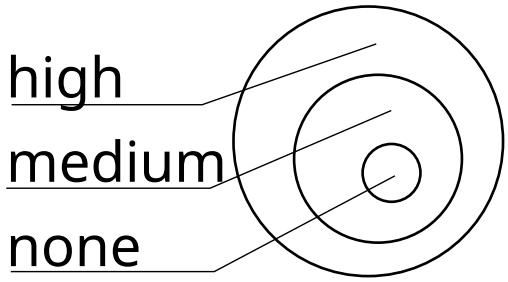


Figure 33: Sets of interference contained in each other.

Table 6 summarizes the availability of server-controlled metadata for the different storage services from this report.

8.2. Interference

Intuitively, Interference refers to the effect that data acquisition has on the data itself. Forensic data acquisition is usually intented as *read-only* access to the data to protect its integrity, but as the results of experiments in the individual chapters show, the effects of read-only access differ considerably within the different technologies.

Despite the complex activities that arise when accessing data over the network, the effects of data access using SMB are minimal, i.e., not even data in the server appears to be changed by read access. Looking at the externally visible behavior, this is compareable to what happens in Nextcloud. However, in Nextcloud the server logs all client accesses, but this information is not accessible by the client. This is different in the business versions of Dropbox and OneDrive, where such data is visible to the client. In this sense, data access visibly interferes with the data on the server, but no information is overwritten. The amount of data accessible by the client is a strict superset of what was accessible before.

We attempt to formalize this behavior. We denote by $f_t(x)$ the set of data objects returned by an API call on a specific storage service at time t to resources identifed by resource identifier x. Intuitively, x can be a file or directory name or any other precise specification of what is to be acquired. We assume that f returns the file contents and all metadata that the service provides.

We now use this notation to define Interference: Assuming two subsequent read accesses to storage resource x at times t_1 and t_2 without any write accesses inbetween, we can distinguish three different cases:

- (high) Anything may change or stay identical. Default case, always true. Formally:

$$\text{high} = \top$$

An example for *high* is cached data. If data is kept in or removed from cache by external choice, without write operations, the return value may change for subsequent calls to the same resource identifier.

- (medium) Data may be extended, may stay the same, but may never be reduced. As long as the first time of access t_1 is earlier than or identical to the second time t_2, returned data of the first call will always be included in the second call. Formally:

$$\text{medium} = \forall x : f(t_1, x) \leq f(t_2, x); t_1 \leq t_2$$

For *medium*, services with read log data are an example. While data contained in a previous call will always be returned, the last read access operation will have created an additional log entry which will be returned for the same call at a later date.

- (none) Data must be identical. No matter the times t and t', returned data is identical. Formally:

$$\text{none} = \forall x : f(t, x) = f(t', x)$$

For *none*, a version control system like Git is a prime example. After commits are published, they may (considering no write access) never change and will thus always return the same data if requested with the same identifier.

Note that the above definition considers only data that is visible to the client. For example, Nextcloud would be classified into the first category despite log data being changed in the server.

We present a classification of the cloud technologies using this dimension, as depicted in Table 7.

None. The services Samba, Nextcloud and Dropbox Basic shall be classified as none. Considering no write access happens between forensic read operations, no data is changed. As no access log data is available, the read operations themselves will not change or add data. Another example for a classification of none is version control systems like Git or SVN, and also backups which are designed to be strictly read only after they are created.

Medium. The services Dropbox Business, Google Business and OneDrive Business shall be classified as medium. These services provide access log data, leading to the addition of data after forensic read operations. No log data is overwritten, but only appended.

High. The services Google Drive Basic, OneDrive Basic and iCloud shall be classified as high. While these provide information on the last read time, there are no logs. Thus, every forensic read operation will overwrite the last read information and thus change data.

8.3. Auditability

We now turn to the notion of Auditability that was informally defined in the introduction and investigated for the different network storage technologies. Looking at the results of the analyses, there are three different behaviors that increasingly support the verification of data accesses that were performed in the past.

The first category considers services that offer no audit information whatsoever. In the best case, assuming no interference, data access can be used to verify that a certain file content existed in the past, but only if the file content has not been changed in the meantime. For example, in Samba/SMB, if there are write accesses to a file between two read accesses, the second read access only retrieves the modified contents. The contents before the write operation are lost. Consequently any claim that a particular version of a file existed in the past requires trust in the investigator, as the storage service itself does not support the verification of this claim in any way.

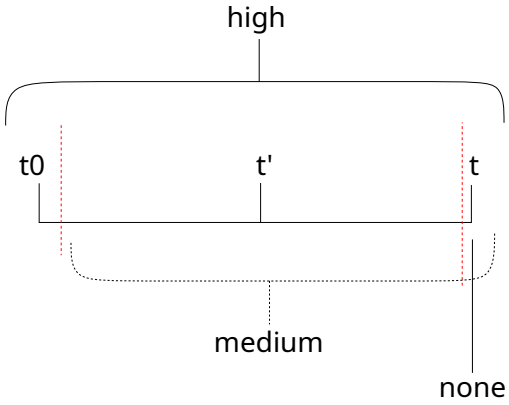


Figure 34: Auditability as influenced by time of (previous) access.

Ideally, a storage service could by itself reproduce data access that was performed by forensic investigators in the past. So if investigators claim that a certain version of a file was acquired at some point in time in the past, the storage service can be independently queried (using valid access credentials) and the service would provide evidence that can be compared to what was claimed to be present at that time. In this case, the storage service acts as a trusted auditor of the actions of investigators. In a sense, this is compareable to accessing the original storage volume in traditional disk forensics to audit the version of a file claimed to be in existence on the drive. We call this category of auditability *high*.

Clearly, for high auditability, the storage service must maintain all previous versions of files, and it must allow to precisely identify particular file versions via the API. As the analyses of the storage services above show, this is not the case for all storage services. For example, the basic version of Dropbox can identify that changes were made to a particular file in the past, but the concrete version is not retrievable anymore. To distinguish this case from the case where no such information is provided by the service, we call this category of auditability *low*.

To summarize, we distinguish three types of Auditability. Assuming read function $f(t, x)$ defined for Interference, we define another function $g(t, x, \tilde{t})$ which takes a second time argument $\tilde{t}$ that may be different to the current time t at the time of read. Depending on the state of auditability, we differentiate three cases:

- (none) No previous resource state may be permanently re-read at a later time. Always true. Formally:

$$\text{none} = \top = \forall \tilde{t} : t_0 \leq \tilde{t} < t : g(t, x, \tilde{t}) \neq f(\tilde{t}, x) \vee \text{high} \vee \text{medium}$$

  An example for *none* may be read-once data that is destroyed after reading. While every call to read may return data, subsequent reads to data may never reproduce previously returned data.
- (medium) Some (not none, not all) previous resource states may be read permanently. Formally:

$$\text{medium} = \forall x, \forall t : \exists \tilde{t} : t_0 \leq \tilde{t} < t : g(t, x, \tilde{t}) \neq f(\tilde{t}, x) \wedge \exists \hat{t} : t_0 \leq \hat{t} < t : g(t, x, \hat{t}) \neq f(\hat{t}, x)$$

  For *medium*, a backup system may be apt. While data is generally preserved permanently, regular deletions of incremental backups may occur. While major backup points, e.g. weekly, yearly, may be preserved indefinately, daily or hourly backups may be deleted and will thus be unreadable.
- (high) All previous resource states may be read permanently. For any times t and $\tilde{t}$, a call to the version-aware function g at call time t with respect to the previous time $\tilde{t}$ will yield the same result as the original read function f did at time $\tilde{t}$. Formally:

$$\text{high} = \forall x, \forall t, \forall \tilde{t} \leq t : g(t, x, \tilde{t}) = f(\tilde{t}, x)$$

Interference	none	Samba; Nextcloud; Dropbox (basic)	Incremental Backups	Git; SVN
	medium		Dropbox (business); Google Drive (business); OneDrive (business)	
	high	Google Drive (basic); OneDrive (basic); iCloud		
		none	medium	high
			Auditability	

Table 7: Overview on the interference and auditability of the storage providers. The rating is detailed in Sections 8.2 and 8.3.

For *high*, a version control system like Git is a prime example. Every file, together with commit and thus time information, may be requested at any later date and will be delivered identically to a time-unaware function at the request time.

We present a classification of the cloud technologies using this dimension, as depicted in Table 7.

None. The services Samba, Nextcloud, Dropbox Basic, Google Drive Basic, OneDrive Basic and iCloud shall be classified as none, as there is no way of accessing previous versions of files indefinitely.

Medium. The services Dropbox Business, Google Drive Business and OneDrive Business shall be classified as medium. Although older versions are available at a later date, the retention of old versions is limited and thus there may be individual versions that are unavailable at a later date. Another example of medium rating is incremental backups. While certail frequencies like weekly or montly backups may be kept indefinitely, incremental daily backups may be deleted and are thus not available anymore.

High. None of the services shall be classified as high. As a comparative example, version control systems like Git or SVN shall be classified as high, as these provide indefinite access to any old versions, provided they are not manually deleted.

8.4. Summary

Given the classifications described above, the network storage services investigated in this report can be classified as shown in Table 7. While it is clear that Auditability is a property that is costly to achieve (because it involves non-negligible amount of additional storage), low or even no Interference also appear to comes with some cost.

Note that the data that is offered by the storage service regarding Auditability may not include any data from logs since otherwise Interference and Auditability would not be fully independent properties. If Interference would also refer to log data, no Interference (i.e., all data including all metadata returned at two subsequent read accesses is exactly identical) would imply no Auditability because for low Auditability also the read accesses should be logged.

9. Conclusion

We presented a comparison of metadata availability and types for six cloud storage providers. The results showed that both interference and auditability differ substantially across the providers.

Considering the value of forensic investigations, this directly leads to more and less suitable points of access. The abstraction layer provided by API access, compared to direct filesystem access, imposes challenges like the need to trust service providers, but also creates opportunities like having consistent states and versions of metadata available.

As this field of forensic computing is still rather new, numerous future research challenges exist. Apart from the services analyzed in this report, other file-based or file-oriented services may be analyzed. Going further, rich cloud platforms like Google Docs or Office 365 may further increase the previously mentioned challenges and opportunities, as they provide an even stronger abstraction layer.

Acknowledgements

We wish to thank Vassil Roussev for supporting this project.

Appendix

The following appendices are accumulated from individial contributions. The first three appendices originate from OneDrive, the fourth from iCloud.

1. OneDrive: JSON representation of resources

Code Listing 1: Drive

```
{
    "id": "string",
    "createdBy": { "@odata.type": "microsoft.graph.identitySet" },
    "createdDateTime": "string (timestamp)",
    "description": "string",
    "driveType": "personal | business | documentLibrary",
    "following": [{"@odata.type": "microsoft.graph.driveItem"}],
    "items": [ { "@odata.type": "microsoft.graph.driveItem" } ],
    "lastModifiedBy": { "@odata.type": "microsoft.graph.identitySet" },
    "lastModifiedDateTime": "string (timestamp)",
    "name": "string",
    "owner": { "@odata.type": "microsoft.graph.identitySet" },
    "quota": { "@odata.type": "microsoft.graph.quota" },
    "root": { "@odata.type": "microsoft.graph.driveItem" },
    "sharepointIds": { "@odata.type": "microsoft.graph.sharepointIds" }
        ,
    "special": [ { "@odata.type": "microsoft.graph.driveItem" }],
    "system": { "@odata.type": "microsoft.graph.systemFacet" },
    "webUrl": "url"
}
```

Code Listing 2: DriveItem

```
{
    "audio": { "@odata.type": "microsoft.graph.audio" },
    "content": { "@odata.type": "Edm.Stream" },
    "cTag": "string (etag)",
    "deleted": { "@odata.type": "microsoft.graph.deleted"},
    "description": "string",
    "file": { "@odata.type": "microsoft.graph.file" },
    "fileSystemInfo": { "@odata.type": "microsoft.graph.fileSystemInfo"
        },
    "folder": { "@odata.type": "microsoft.graph.folder" },
    "image": { "@odata.type": "microsoft.graph.image" },
    "location": { "@odata.type": "microsoft.graph.geoCoordinates" },
    "malware": { "@odata.type": "microsoft.graph.malware" },
    "package": { "@odata.type": "microsoft.graph.package" },
    "pendingOperations": { "@odata.type": "microsoft.graph.
       pendingOperations" },
    "photo": { "@odata.type": "microsoft.graph.photo" },
    "publication": {"@odata.type": "microsoft.graph.publicationFacet"},
    "remoteItem": { "@odata.type": "microsoft.graph.remoteItem" },
    "root": { "@odata.type": "microsoft.graph.root" },
    "searchResult": { "@odata.type": "microsoft.graph.searchResult" },
    "shared": { "@odata.type": "microsoft.graph.shared" },
    "sharepointIds": { "@odata.type": "microsoft.graph.sharepointIds" }
       ,
    "size": 1024,
    "specialFolder": { "@odata.type": "microsoft.graph.specialFolder" }
       ,
    "video": { "@odata.type": "microsoft.graph.video" },
    "webDavUrl": "string",
     /* relationships */
    "activities": [{"@odata.type": "microsoft.graph.itemActivity"}],
    "analytics": {"@odata.type": "microsoft.graph.itemAnalytics"},
    "children": [{ "@odata.type": "microsoft.graph.driveItem" }],
    "createdByUser": { "@odata.type": "microsoft.graph.user" },
    "lastModifiedByUser": { "@odata.type": "microsoft.graph.user" },
    "permissions": [ {"@odata.type": "microsoft.graph.permission"} ],
    "subscriptions": [ {"@odata.type": "microsoft.graph.subscription"}
       ],
    "thumbnails": [ {"@odata.type": "microsoft.graph.thumbnailSet"}],
    "versions": [ {"@odata.type": "microsoft.graph.driveItemVersion"}],

    /* inherited from baseItem */
    "id": "string (identifier)",
    "createdBy": {"@odata.type": "microsoft.graph.identitySet"},
    "createdDateTime": "String (timestamp)",
    "eTag": "string",
```

```
    "lastModifiedBy": {"@odata.type": "microsoft.graph.identitySet"},
    "lastModifiedDateTime": "String (timestamp)",
    "name": "string",
    "parentReference": {"@odata.type": "microsoft.graph.itemReference"},
    "webUrl": "string",

    /* instance annotations */
    "@microsoft.graph.conflictBehavior": "string",
    "@microsoft.graph.downloadUrl": "url",
    "@microsoft.graph.sourceUrl": "url"
}
```

Code Listing 3: versions.json

```
{
    "content": { "@odata.type": "Edm.Stream" },
    "id": "string",
    "lastModifiedBy": { "@odata.type": "microsoft.graph.identitySet" },
    "lastModifiedDateTime": "2016-01-01T15:20:01.125Z",
    "publication": { "@odata.type": "microsoft.graph.publicationFacet"
       },
    "size": 12356,

    /* instance annotations */
    "@microsoft.graph.downloadUrl": "url",
}
```

Code Listing 4: Activity

```
{
    "appActivityId": "String",
    "activitySourceHost": "String (host name/domain/URL)",
    "userTimezone": "String",
    "appDisplayName": "String",
    "activationUrl": "String (URL)",
    "contentUrl": "String (URL)",
    "fallbackUrl": "String (URL)",
    "createdDateTime": "DateTimeOffset",
    "lastModifiedDateTime": "DateTimeOffset",
    "expirationDateTime": "DateTimeOffset",
    "id": "String",
    "status": "active | updated | deleted | ignored",
    "contentInfo": { "@odata.type": "microsoft.graph.Json" },
    "visualElements": { "@odata.type": "microsoft.graph.visualInfo" },
    "historyItems": [{ "@odata.type": "microsoft.graph.
       activityHistoryItem" }]
}
```

name	path	sha-256-sum
foo.txt	root	2d2da19605a34e037dbe82173f98a992a530a5fdd53dad882f570d4ba204ef30
IMG_20200217_160600.jpg	root	bbf9befa1f534f782b2c001f39d6501c4c0898cce47c2dd40f96996a9d4b4198
Summieren.xlsx (Version 1)	root	0d5a9c3eb2acd00691bd92f81db208ee6dc27bd953d4f3e7d9192f93a85aacc4
Summieren.xlsx (Version 2)	root	06f0f057b58a49735cef3cb5aeeaf445eb8e0cd60aa47b114fba923e1e0a9989
test.docx	root	12b30d143791e0cb0454de942b10b85faa9a938e021208694af340d467142b11
test.pdf	root	f73d0839536a4cee0cb6ffeaf91f08c735736b148967e583844f7b65d6c528eb
20200524_152059000_iOS.jpg	root/Bilder	92d7e3d2926e1802a331a4bbddde5cfceb57e9c513e27e9b1bcacbf4160e3675
IMG_20200217_131005.jpg	root/Bilder	133d26b1002a8240159908c70ea1447339fc1b507f631e6e53b21acc1c6e6885

Table 8: Uploaded files

Code Listing 5: permissions.json

```
{
    "id": "string (identifier)",
    "grantedTo": {"@odata.type": "microsoft.graph.identitySet"},
    "grantedToIdentities": [{"@odata.type": "microsoft.graph.
       identitySet"}],
    "grantedToV2": {"@odata.type": "microsoft.graph.
       sharePointIdentitySet"},
    "grantedToIdentitiesV2": [{"@odata.type": "microsoft.graph.
       sharePointIdentitySet"}],
    "inheritedFrom": {"@odata.type": "microsoft.graph.itemReference"},
    "invitation": {"@odata.type": "microsoft.graph.sharingInvitation"},
    "link": {"@odata.type": "microsoft.graph.sharingLink"},
    "roles": ["string"],
    "shareId": "string",
    "expirationDateTime": "string (timestamp)",
    "hasPassword": "boolean"
}
```

2. OneDrive: Uploaded files

2.1. File tree of OneDrive's test content. `root`

```
├── bigfile.bin
├── Erste Schritte mit OneDrive.pdf
├── foo.txt
├── IMG_20200217_160600.jpg
├── Summieren.xlsx
├── test.docx
├── test.pdf
├── test.txt
├── Bilder
│   ├── 20200524_152059000_iOS.jpg
│   └── IMG_20200217_131005.jpg
└── Dokumente
```

3. OneDrive: Results

3.1. Metadata.

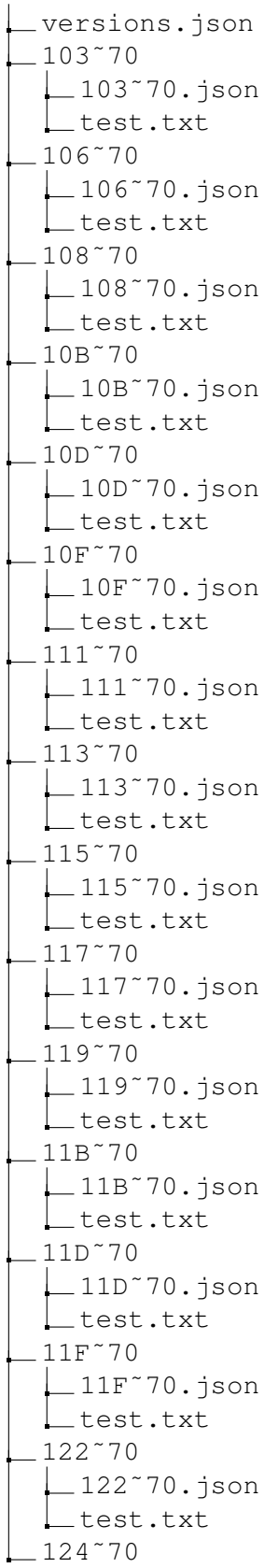
Code Listing 6: sha256 sums and MAC-times of files before uploading to OneDrive (and after Downloading)

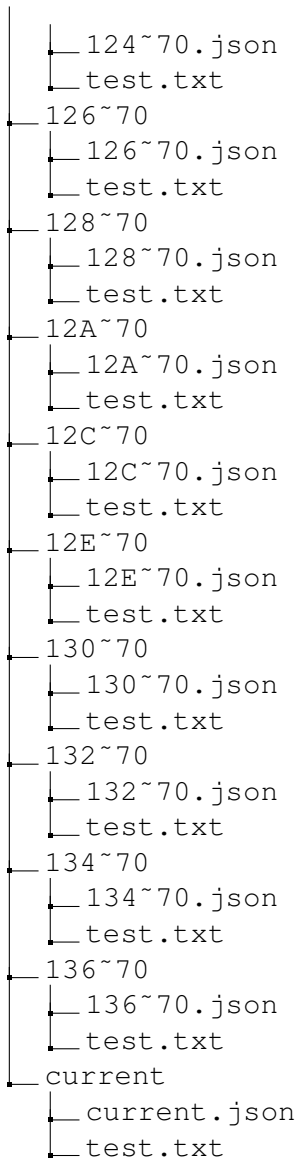
```
20200524_152059000_iOS.jpg
        sha256:        92D7E3D2926E1802A331A4BBDDDE5CFCEB57E9C513E27E9
           B1BCACBF4160E3675        92d7e3d2926e1802a331a4bbddde5cfceb
           57e9c513e27e9b1bcacbf4160e3675
        last modified: 2021-12-09 11:38:46     2020-05-24 17:20:59:
               not equal
        creation time stamp:   2021-12-09 11:38:46     2022-02-02 12:4
           1:16.811814:     not equal
        recent access time - Original Data:    2022-02-02 13:26:51.687
           908

IMG_20200217_131005.jpg
        sha256:        133D26B1002A8240159908C70EA1447339FC1B507F631E6
           E53B21ACC1C6E6885        133d26b1002a8240159908c70ea1447339
           fc1b507f631e6e53b21acc1c6e6885
        last modified: 2021-12-09 11:38:47     2020-02-17 13:10:06:
               not equal
        creation time stamp:   2021-12-09 11:38:47     2022-02-02 12:4
           1:16.817677:     not equal
        recent access time - Original Data:    2022-02-02 13:26:51.691
           816

test.pdf
        sha256:        F73D0839536A4CEE0CB6FFEAF91F08C735736B148967E58
           3844F7B65D6C528EB        f73d0839536a4cee0cb6ffeaf91f08c735
           736b148967e583844f7b65d6c528eb
        last modified: 2021-12-09 11:39:30     2021-12-07 16:56:04:
               not equal
        creation time stamp:   2021-12-09 11:39:30     2022-02-02 12:4
           1:16.806928:     not equal
        recent access time - Original Data:    2022-02-02 13:26:51.685
           953

Summieren.xlsx
        sha256:        06F0F057B58A49735CEF3CB5AEEAF445EB8E0CD60AA47B1
           14FBA923E1E0A9989        06f0f057b58a49735cef3cb5aeeaf445eb
           8e0cd60aa47b114fba923e1e0a9989
        last modified: 2022-01-24 13:35:36     2022-01-24 14:33:44:
               not equal
        creation time stamp:   2022-01-24 13:35:36     2022-02-02 12:4
           1:16.793248:     not equal
```

```
23          recent access time - Original Data:    2022-02-02 13:26:51.684
               976
24
25 test.docx
26          sha256:        12B30D143791E0CB0454DE942B10B85FAA9A938E0212086
               94AF340D467142B11        12b30d143791e0cb0454de942b10b85faa
               9a938e021208694af340d467142b11
27          last modified: 2022-01-17 17:59:19     2021-12-07 16:55:40:
                  not equal
28          creation time stamp:   2022-01-17 17:59:19     2022-02-02 12:4
               1:16.802043:     not equal
29          recent access time - Original Data:    2022-02-02 13:26:51.685
               953
30
31 foo.txt
32          sha256:        2D2DA19605A34E037DBE82173F98A992A530A5FDD53DAD8
               82F570D4BA204EF30        2d2da19605a34e037dbe82173f98a992a5
               30a5fdd53dad882f570d4ba204ef30
33          last modified: 2022-01-17 17:59:20     2021-12-07 16:45:25:
                  not equal
34          creation time stamp:   2022-01-17 17:59:20     2022-02-02 12:4
               1:16.770773:     not equal
35          recent access time - Original Data:    2022-02-02 13:26:51.673
               250
36
37 IMG_20200217_160600.jpg
38          sha256:        BBF9BEFA1F534F782B2C001F39D6501C4C0898CCE47C2DD
               40F96996A9D4B4198        bbf9befa1f534f782b2c001f39d6501c4c
               0898cce47c2dd40f96996a9d4b4198
39          last modified: 2022-01-17 17:59:21     2020-02-17 16:06:01:
                  not equal
40          creation time stamp:   2022-01-17 17:59:21     2022-02-02 12:4
               1:16.771751:     not equal
41          recent access time - Original Data:    2022-02-02 13:26:51.677
               159
```

3.2. Delta file tree structure.

```
delta
└── e0627dfe142cb9a5
    ├── E0627DFE142CB9A5!101
    │   ├── item.json
    │   └── permissions.json
    └── E0627DFE142CB9A5!123
        ├── item.json
        ├── permissions.json
        ├── test.txt
        └── versions
```


versions.json
103~70
103~70.json
test.txt
106~70
106~70.json
test.txt
108~70
108~70.json
test.txt
10B~70
10B~70.json
test.txt
10D~70
10D~70.json
test.txt
10F~70
10F~70.json
test.txt
111~70
111~70.json
test.txt
113~70
113~70.json
test.txt
115~70
115~70.json
test.txt
117~70
117~70.json
test.txt
119~70
119~70.json
test.txt
11B~70
11B~70.json
test.txt
11D~70
11D~70.json
test.txt
11F~70
11F~70.json
test.txt
122~70
122~70.json
test.txt
124~70



4. iCloud: Detailed Endpoint description

4.1. POST /retrieveItemDetailsInFolders.

Request URL: https://p57-drivews.icloud.com/retrieveItemDetailsInFolders

Description. Retrieves meta data as well as the contents of a folder in iCloud Drive for a given folder identifier.

Parameters.

Name	Type	Example
appIdentifier	String	iclouddrive
reqIdentifier	String	88ffc43b-ad46-4a5d-8044-2e312f69201a
clientBuildNumber	String	2206Project45
clientMasteringNumber	String	2206B40
clientId	String	efeecc43-9363-41e8-a4db-238a29f128a
dsid	String	1002654000

Request Body.

```
[
  {
    "drivewsid": "FOLDER::com.apple.CloudDocs::root",
    "partialData": true,
    "includeHierarchy": true
  }
]
```

Response.

```
[
  {
    "dateCreated": "2014-10-27T17:09:55Z",
    "drivewsid": "FOLDER::com.apple.CloudDocs::root",
    "docwsid": "root",
    "zone": "com.apple.CloudDocs",
    "name": "",
    "hierarchy": [],
    "etag": "52do",
    "type": "FOLDER",
    "assetQuota": 20071568166,
    "fileCount": 10698,
    "shareCount": 3,
    "shareAliasCount": 0,
    "directChildrenCount": 136,
    "items": [
      {
        "drivewsid": "FILE::com.apple.CloudDocs::BBAD13A7-0243-42C1-B684-37946D4A48A1",
        "docwsid": "BBAD13A7-0243-42C1-B684-37946D4A48A1",
        "etag": "3kf1::3keu"
      },
      {
        "drivewsid": "FILE::com.apple.CloudDocs::A536CE29-189A-4637-BFE6-883743C22FCC",
        "docwsid": "A536CE29-189A-4637-BFE6-883743C22FCC",
        "etag": "19gq::19gn"
      }],
```

```
    "numberOfItems": 27,
    "status": "OK"
  }
]
```

4.2. POST /retrieveTrashDetails.

Request URL: https://p57-drivews.icloud.com/retrieveTrashDetails

Description. Retrieves information about the iCloud trash folder. With the parameters and the request body that were observed in the iCloud web interface, the response seems to contain only the number of elements in the trash folder.

Parameters.

Name	Type	Example
appIdentifier	String	iclouddrive
reqIdentifier	String	862803e5-ea7a-43a2-b294-c26aa7908673
clientBuildNumber	String	2206Project45
clientMasteringNumber	String	2206B40B40
clientId	String	efeecc43-9363-41e8-a4db-238a29f128a
dsid	String	1002654000

Request Body.

```
{
  "includeShallowCount": true
}
```

Response.

```
{
        "shallowCount":2
}
```

4.3. POST /storageUsageInfo.

Request URL: https://setup.icloud.com/setup/ws/1/storageUsageInfo

Description. Retrieves information about the storage usage in iCloud Drive. In particular provides information about the distribution of the used storage space with respect to different data categories (photos, backups, generic documents, e-mails).

Parameters.

Name	Type	Example
clientBuildNumber	String	2206Project45
clientMasteringNumber	String	2206B40
clientId	String	48e053be-aeb5-435e-83a2-9e1281202d56
dsid	String	8387056927

Request Body. null

Response.


```
{
  "storageUsageByMedia": [
    {
      "mediaKey": "photos",
      "displayLabel": "Fotos und Videos",
      "displayColor": "ffcc00",
      "usageInBytes": 131628665367
    },
    {
      "mediaKey": "backup",
      "displayLabel": "Backups",
      "displayColor": "5856d6",
      "usageInBytes": 25215480933
    },
    {
      "mediaKey": "docs",
      "displayLabel": "Dokumente",
      "displayColor": "ff9500",
      "usageInBytes": 21126233644
    },
    {
      "mediaKey": "mail",
      "displayLabel": "Mail",
      "displayColor": "007aff",
      "usageInBytes": 1791602084
    }
  ],
  "storageUsageInfo": {
    "compStorageInBytes": 0,
    "usedStorageInBytes": 179786467578,
```

```
    "totalStorageInBytes": 214748364800,
    "commerceStorageInBytes": 214748364800
  },
  "quotaStatus": {
    "overQuota": false,
    "haveMaxQuotaTier": false,
    "almost-full": false,
    "paidQuota": true
  }
}
```

4.4. POST /download/batch.

Request URL: https://p57-docws.icloud.com/ws/com.apple.CloudDocs/download/batch

Description. Retrieves file information for a given array of document_id's. The returning information consists of several attributes which are used by the icloud web interface to directly open a file in the browser. Examples are an unique ownerId as well as a url to the specific file.

Parameters.

Name	Type	Example
token	String	=Ew==BST_IAAAAAAABLwIAAAAAGH0_84RDmdzLmlj bG91ZC5hdXRovQDBw9DDY-Utd0b5Yvny9nwPbwf4t19EOPk _BJQhOE1qneCscSmQk0_UzGHtGM6ew4_CfgK75TxB _G-lWi-SgbgVliAapK5Me9iJAW_TjH6L88TRDq4pkc fGdMuUFRsNkNy1rMInyaMD7-wsoX2ijWaUhIA01g
appIdentifier	String	iclouddrive
reqIdentifier	String	64d200c2-6f4d-45ae-9be2-51766a9c59b6
clientBuildNumber	String	2206Project45
clientMasteringNumber	String	2206B40B40
clientId	String	efeecc43-9363-41e8-a4db-238a29f128a
dsid	String	1002654000

Request Body.

```
[
  {
    "document\_id": "BBAD13A7-0243-42C1-B684-37946D4A48A1"
  },
  {
    "document\_id": "A536CE29-189A-4637-BFE6-883743C22FCC"
  }
]
```

Response.

```
[
  {
    "document\_id": "BBAD13A7-0243-42C1-B684-37946D4A48A1",
    "owner\_dsid": 1002654000,
    "data\_token": {
      "url": "https://cvws.icloud-content.com/B/AR\_aL9dDzK72ETk\_DgW-
         PSk4\_x20AVyAtE5tUjMQ3US3Os56iCC3COv\_/5teamDouble.pdf?o=ApXk7
         c-8anMu1bd1nqL2MSWGgbmPWeg4YBHfkT-kPikr\&v=1\&x=3\&a=CAog2
         QaVPkEYTJMtapcCk7YBuC8zJCuGqt1IwEY1-F7yaysSbRDvkp2o6i8Yj4
         rUqOovIgEAUgQ4\_x20WgS3COv\_aibXieuxn2MOGme0GZQa-XIVDcHRzTE1
         VxTVz1G3VI1LzFcTb\_gLH3ImimP6E9i0om0l-Cpdo9E2gFOKddSo\_11IKD8
         OkGoa8BCagzn4flg\&e=1643447125\&fl=\&r=8fd7556d-eaf0-4951-b9cd
         -b4168411fb4f-1\&k=wQ14eApDUbWWnP8rEGiSSg\&ckc=com.apple.
```

```
        clouddocs\&ckz=com.apple.CloudDocs\&p=57\&s=Pa36bIoql9NPw5i53v
        0YTT0mvfs",
      "token": "CAogyhPGt4xbaBVrT+EFFKY8M0CTD1CDzlFnLSvF7IcdpOUSbRDwkp2
        o6i8YkIrUqOovIgEAUgQ4/x20WgS3COv/aiYthhu83i5WmlA02k5iErTpUSewi
        3Ytn51h78U2JVUzWhjEaY2AcHImlgolwOAQO6guM6czHU8HOKtR3r+
        EvxqZCGmLJxwaSxQEwPxTBuw=",
      "signature": "AR/aL9dDzK72ETk/DgW+PSk4/x20",
      "wrapping\_key": "wQ14eApDUbWWnP8rEGiSSg==",
      "reference\_signature": "AVyAtE5tUjMQ3US3Os56iCC3COv/"
    },
    "double\_etag": "3kf1::3keu"
  },
  {
    "document\_id": "A536CE29-189A-4637-BFE6-883743C22FCC",
    "owner\_dsid": 1002654000,
    "data\_token": {
      "url": "https://cvws.icloud-content.com/B/AdoAaTujk\_At1J325RF5-
        gb9byVSAQoD2Zy-VHSCmvGUdMiH-XHeEUpk/abschluss60336d1b-d1cc-46
        cd-a466-c598727c1a79.pdf?o=AkxE0gKn-ZVshk1mwUT\_v12uHPCgj7
        PboeTqoFXrObpd\&v=1\&x=3\&a=CAog9Qyn3Acs4tM\_0h0mXoTxuGzss-
        siJbpv4gQ\_JAauHCkSbRD7kp2o6i8Ym4rUqOovIgEAUgT9
        byVSWgTeEUpkaiZUKsKuT3fRXb75crOZHxDNXpLSn\_rWByUCv1H4xNT1JR7
        lpOy1LHImdlbayL\_ypfuLRqtkrkyuorLixMO-P4suPp2WynFzLJpeTeeAGUg
        \&e=1643447125\&fl=\&r=8fd7556d-eaf0-4951-b9cd-b4168411fb4f-1
        \&k=WlEpRbOKO-7FFCBg39wpMg\&ckc=com.apple.clouddocs\&ckz=com.
        apple.CloudDocs\&p=57\&s=CEuqADAT5urk7hE\_70-GFBjXAxE",
      "token": "CAogJGjvSIp9b+Q1pNjjTK/+2HUFck2BFgvOu6HYSHTD2EMSbRD7kp2
        o6i8Ym4rUqOovIgEAUgT9byVSWgTeEUpkaibd6bnLBNdw2d9QrZC7M09asYBBz
        8OGyZdOqBFKEYtwwYPrDmrxhnImhxDM1oOYnJEFB0JMzQPYhLZrjeKVj+k0A9
        QbHB1mS+uXHJhngs4=",
      "signature": "AdoAaTujk/At1J325RF5+gb9byVS",
      "wrapping\_key": "WlEpRbOKO+7FFCBg39wpMg==",
      "reference\_signature": "AQoD2Zy+VHSCmvGUdMiH+XHeEUpk"
    },
    "double\_etag": "19gq::19gn"
  }
]
```


References

[1] WIKIPEDIA (Hrsg.): *OneDrive*. 2021. – URL https://en.wikipedia.org/w/index.php?title=OneDrive&oldid=1061197858. – Zugriffsdatum: 29.01.2022

[2] BOREAN, Jordan: *smbprotocol*. github. 2021. – URL https://github.com/jborean93/smbprotocol. – Zugriffsdatum: 2022-02-08

[3] BREITINGER, Frank ; ZHANG, Xiaolu ; QUICK, Darren: A forensic analysis of rclone and rclone's prospects for digital forensic investigations of cloud storage. In: *Proceedings DFRWS APAC*, September 2022

[4] CARRIER, Brian: *File system forensic analysis*. Addison-Wesley Professional, 2005

[5] CASEY, Eoghan: *Digital Evidence and Computer Crime - Forensic Science, Computers and the Internet, 3rd Edition*. Academic Press, 2011. – URL http://www.elsevierdirect.com/product.jsp?isbn=9780123742681. – ISBN 978-0-12-374268-1

[6] CHOO, Kim-Kwang R. ; ESPOSITO, Christian ; CASTIGLIONE, Aniello: Evidence and Forensics in the Cloud: Challenges and Future Research Directions. In: *IEEE Cloud Computing* 4 (2017), Nr. 3, S. 14–19

[7] COULTER, David ; SATRAN, Michael ; BATCHELOR, Drew: *Microsoft SMB protocol and CIFS Protocol Overview*. https://docs.microsoft.com/en-us/windows/win32/fileio/microsoft-smb-protocol-and-cifs-protocol-overview. Januar 2021. – part of Windows 2000 Web and Application Services Technical Overview

[8] CRAIG-WOOD, Nick: *Rclone*. https://rclone.org/. Juli 2022

[9] DANNER, Merlin ; BOREAN, Jordan: *smbprotocol-readonly*. gitlab.cs.fau.de. 2022. – URL https://gitlab.cs.fau.de/yp98ocaw/smbprotocol-readonly. – Zugriffsdatum: 2022-02-08

[10] DROPBOX: *Dropbox for HTTP Developers*. https://www.dropbox.com/developers/documentation/http/documentation. – [Online; accessed 03-February-2022]

[11] DROPBOX: *Getting started with the Dropbox API*. https://www.dropbox.com/developers/reference/getting-started. – [Online; accessed 09-February-2022]

[12] FOUNDATION, Python S.: *argparse - Python Standard Library*. – URL https://docs.python.org/3/library/argparse.html. – Zugriffsdatum: 2022-02-08

[13] FRÖWIS, Michael ; GOTTSCHALK, Thilo ; HASLHOFER, Bernhard ; RÜCKERT, Christian ; PESCH, Paulina: Safeguarding the evidential value of forensic cryptocurrency investigations. In: *Digit. Investig.* 33 (2020), S. 200902. – URL https://doi.org/10.1016/j.fsidi.2019.200902

[14] HAYNES, T. ; NOVECK, D.: *Network File System (NFS) Version 4 Protocol*. IETF Request for Comments 7530. März 2015

[15] JAMES, Stuart H. (Hrsg.) ; NORDBY, Jon J. (Hrsg.): *Forensic Science. An Introduction to Scientific and Investigative Techniques*. Third. CRC press, 2009

[16] KANIA, Stefan: *Samba 4 Das Handbuch für Administratoren*. Carl Hanser Verlag, 2021. – ISBN 978-3-446-46977-8

[17] MANRAL, Bharat ; SOMANI, Gaurav ; CHOO, Kim-Kwang R. ; CONTI, Mauro ; GAUR, Manoj S.: A Systematic Survey on Cloud Forensics Challenges, Solutions, and Future Directions. In: *ACM Comput. Surv.* 52 (2019), nov, Nr. 6. – URL https://doi.org/10.1145/3361216. – ISSN 0360-0300

[18] MICROSOFT: [MS-SMB2]: Server Message Block (SMB) Protocol Versions 2 and 3. . – URL https://msdn.microsoft.com/en-us/library/cc246482.aspx. – Zugriffsdatum: 2022-02-08

[19] MICROSOFT: *Overview of Microsoft Graph - Microsoft Graph*. 05.02.2022. – URL https://docs.microsoft.com/en-us/graph/overview?view=graph-rest-1.0. – Zugriffsdatum: 05.02.2022

[20] MICROSOFT: *How OneDrive safeguards your data in the cloud*. 29.01.2022. – URL https://support.microsoft.com/en-us/office/how-onedrive-safeguards-your-data-in-the-cloud-23c6ea94-3608-48d7-8bf0-80e142edd1e1. – Zugriffsdatum: 29.01.2022

[21] MICROSOFT: *OneDrive system requirements*. 29.01.2022. – URL https://support.microsoft.com/en-us/office/onedrive-system-requirements-cc0cb2b8-f446-445c-9b52-d3c2627d681e. – Zugriffsdatum: 29.01.2022

[22] MUNISWAMY-REDDY, Kiran-Kumar ; SELTZER, Margo I.: Provenance as first class cloud data. In: *ACM SIGOPS Oper. Syst. Rev.* 43 (2009), Nr. 4, S. 11–16. – URL https://doi.org/10.1145/1713254.1713258

[23] ROCHET, Jean-Charles ; TIROLE, Jean: Platform Competition in Two-Sided Markets. In: *Journal of the European Economic Association* 1 (2003), Juni, Nr. 4, S. 990–1029

[24] ROUSSEV, Vassil: *Digital Forensic Science: Issues, Methods, and Challenges*. Morgan & Claypool Publishers, 2016 (Synthesis Lectures on Information Security, Privacy, & Trust). – URL https://doi.org/10.2200/S00738ED1V01Y201610SPT019

[25] ROUSSEV, Vassil ; AHMED, Irfan ; BARRETO, Andres ; MCCULLEY, Shane ; SHANMUGHAN, Vivek: Cloud forensics-Tool development studies & future outlook. In: *Digit. Investig.* 18 (2016), S. 79–95. – URL https://doi.org/10.1016/j.diin.2016.05.001

[26] ROUSSEV, Vassil ; BARRETO, Andres ; AHMED, Irfan: API-Based Forensic Acquisition of Cloud Drives. In: PETERSON, Gilbert (Hrsg.) ; SHENOI, Sujeet (Hrsg.): *Advances in Digital Forensics XII*. Cham : Springer International Publishing, 2016, S. 213–235. – ISBN 978-3-319-46279-0

[27] ROUSSEV, Vassil ; MCCULLEY, Shane: Forensic analysis of cloud-native artifacts. In: *Digital Investigation* 16 (2016), S. S104–S113. – URL https://www.sciencedirect.com/science/article/pii/S174228761630007X. – DFRWS 2016 Europe. – ISSN 1742-2876

[28] SAFERSTEIN, Richard: *Criminalistics. An Introduction to Forensic Science*. Tenth. Pearson, 2011

[29] SAMBA TEAM: *smb.conf — The configuration file for the Samba suite*. – URL https://www.samba.org/samba/docs/current/man-html/smb.conf.5.html

[30] TS, Jay ; ECKSTEIN, Robert ; COLLIER-BROWN, David: *Using Samba*. Beijing Sebastopol, CA : O'Reilly, 2003. – ISBN 9780596002565