API-Based Forensic Acquisition of Cloud Drives

Vassil Roussev, Andres Barreto, Irfan Ahmed

Chapter 11

API-BASED FORENSIC ACQUISITION OF CLOUD DRIVES

Vassil Roussev, Andres Barreto and Irfan Ahmed

Abstract Cloud computing and cloud storage services, in particular, pose new challenges to digital forensic investigations. Currently, evidence acquisition for these services follows the traditional method of collecting artifacts residing on client devices. This approach requires labor-intensive reverse engineering effort and ultimately results in an acquisition that is inherently incomplete. Specifically, it makes the incorrect assumption that all the storage content associated with an account is fully replicated on the client. Additionally, there is no current method for acquiring historical data in the form of document revisions, nor is there a way to acquire cloud-native artifacts from targets such as Google Docs.

This chapter introduces the concept of API-based evidence acquisition for cloud services, which addresses the limitations of traditional acquisition techniques by utilizing the officially-supported APIs of the services. To demonstrate the utility of this approach, a proof-of-concept acquisition tool, `kumodd`, is presented. The `kumodd` tool can acquire evidence from four major cloud drive providers: Google Drive, Microsoft OneDrive, Dropbox and Box. The implementation provides command-line and web user interfaces, and can be readily incorporated in established forensic processes.

Keywords: Cloud forensics, cloud drives, API-based acquisition

1. Introduction

Cloud computing is emerging as the primary model for delivering information technology services to Internet-connected devices. It abstracts away the physical computing and communications infrastructure, and enables customers to effectively rent (instead of own and maintain) as much infrastructure as needed. According to NIST [14], cloud computing has five essential characteristics that distinguish it from previous

service models: (i) on-demand self service; (ii) broad network access; (iii) resource pooling; (iv) rapid elasticity; and (v) measured service.

The underpinning technological development that has made the cloud possible is the massive adoption of virtualization on commodity hardware systems. Ultimately, this allows for a large pool of resources, such as a data center, to be provisioned and load-balanced at a fine granularity, and for the computations of different users (and uses) to be strongly isolated.

The first public cloud services – Amazon Web Services (AWS) – were introduced by Amazon in 2006. According to a 2015 report by RightScale [18], cloud adoption has become ubiquitous: 93% of businesses are experimenting with cloud deployments, with 82% adopting a hybrid strategy that combines the use of multiple providers (usually in a public-private configuration). Nonetheless, much of the technology transition is still ahead, as 68% of enterprises have less than 20% of their application portfolios running in cloud environments. Gartner [8] predicts that another two to five years will be needed before cloud computing reaches the "plateau of productivity" [9], heralding a period of mainstream adoption and widespread productivity gains.

Meanwhile, cloud forensics is in its infancy. Few practical solutions exist for the acquisition and analysis of cloud evidence, and most of them are minor adaptations of traditional methods and tools. Indeed, NIST, the principal standardization body in the United States, is still attempting to build consensus on the challenges involved in performing forensics of cloud data. A recent NIST report [14] identifies 65 separate challenges involved in cloud forensics.

This research focuses on a specific problem – the acquisition of data from cloud storage services. Cloud storage services are extremely popular, with providers such as Google Drive, Microsoft OneDrive, Dropbox and Box offering consumers between 2 GB and 15 GB of free cloud storage. Cloud storage is also widely used by mobile devices to share data across applications that are otherwise isolated from each other. Therefore, a robust evidence acquisition method is a necessity. Additionally, due to the wide variety of cloud storage services and the rapid introduction of new services, evidence acquisition methods and tools should be adaptable and extensible.

In traditional forensic models, an investigator works with physical evidence containers such as storage media or integrated embedded devices such as smartphones. In these scenarios, it is easy to identify the processor that performs the computations as well as the media that store traces of the computations, and to physically collect, preserve and analyze the relevant information content. As a result, research has focused on discov-

ering and acquiring every little piece of log and timestamp information, and extracting every last bit of discarded data that applications and the operating system leave behind.

Conceptually, cloud computing breaks this model in two major ways. First, resources such as CPU cycles, RAM and secondary storage are pooled (e.g., RAID storage) and then allocated at a fine granularity. This results in physical media that usually contain data owned by many users. Additionally, data relevant to a single case can be spread across numerous storage media and (potentially) among different providers responsible for different layers in the cloud stack. Applying the conventional model introduces several procedural, legal and technical problems that are unlikely to have an efficient solution in the general case. Second, computations and storage records are ephemeral because virtual machine (VM) instances are continually created and destroyed and working storage is routinely sanitized.

As discussed in the next section, cloud storage forensics treats the problem as just another instance of application forensics. It applies basic differential analysis techniques [7] to gain an understanding of the artifacts present on client devices by taking before and after snapshots and deducing the relevant cause and effect relationships. During an actual investigation, an analyst would be interpreting the state of the system based on these known relationships.

Unfortunately, there are several problems with the application of existing client-side methods:

- **Completeness:** The reliance on client-side data can exclude critical case data. An example is the selective replication of cloud drive data, which means that a client device may not have a local copy of all the stored data. As usage grows – Google Drive already offers up to 30 TB of storage – this will increasingly be the typical situation.

- **Correctness and Reproducibility:** It is infeasible to reverse engineer all the aspects of an application's functionality without its source code; this immediately calls into question the correctness of the analysis. Furthermore, cloud storage applications on a client are updated frequently with new features introduced on a regular basis. This places a burden on cloud forensics to keep up the reverse engineering efforts, making it harder to maintain the reproducibility of analyses.

- **Cost and Scalability:** Manual client-side analysis is burdensome and does not scale with the rapid growth and the variety of services (and service versions).

This chapter presents an alternative approach for acquiring evidence from cloud storage services by leveraging the official APIs provided by the services. This approach, which eliminates the need for reverse engineering, has the following conceptual advantages:

- APIs are well-documented, official interfaces through which cloud applications on a client communicate with services. They tend to change slowly and any changes are clearly marked; new features may be incorporated incrementally in an acquisition tool.
- It is easy to demonstrate completeness and reproducibility using an API specification.
- Web APIs tend to follow patterns, which makes it possible to adapt existing code to a new (similar) service with modest effort. It is often practical to write an acquisition tool for a completely new service from scratch in a few hours.

To demonstrate the feasibility of the approach and to gain firsthand experience with the acquisition process, a proof-of-concept tool named `kumodd` has been developed. The tool can perform complete (or partial) acquisition of cloud storage account data. It works with four popular services, Google Drive, Microsoft OneDrive, Dropbox and Box, and supports the acquisition of revisions and cloud-only documents. The prototype is written in Python and offers command line and web-based user interfaces.

2. Related Work

This section summarizes essential cloud terminology and discusses related work.

2.1 Cloud Computing

The National Institute of Standards and Technology (NIST) [14] defines cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." With respect to public cloud services – the most common case – this means that the physical hardware on which computations take place is owned and maintained by the provider, and is, thus, part of the deployed software stack. Generally, customers have the option to pay per unit of CPU, storage and network use, although other business arrangements are also possible.

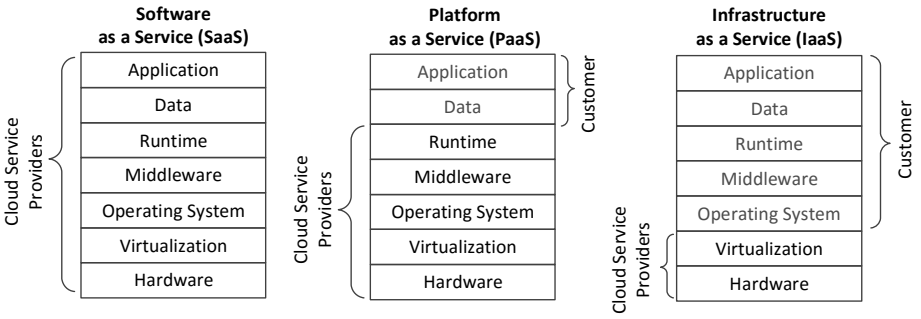


Figure 1. Cloud service models and ownership of layers (public cloud).

Cloud computing services are commonly classified into three canonical models: (i) software as a service (SaaS); (ii) platform as a service (PaaS); and infrastructure as a service (IaaS). In practice, the distinctions are often not clear cut and practical information technology cloud solutions – and potential investigative targets – may incorporate elements of all three canonical models. As illustrated in Figure 1, it is useful to decompose a cloud computing environment into a stack of layers (from low to high): (i) hardware; (ii) virtualization (consisting of a hypervisor that enables the installation of virtual machines); (iii) operating system (installed on each virtual machine); (iv) middleware; (v) runtime environment; (vi) data; and (vii) application.

Depending on the deployment scenario, different layers may be managed by different parties. In a private deployment, the entire stack is hosted by the owner and the overall forensic picture is very similar to that of a non-cloud information technology target. Data ownership is clear, as are the legal and procedural paths to obtain the data; indeed, the very use of the term "cloud" is mostly immaterial to forensics.

In a public deployment, the SaaS/PaaS/IaaS classification becomes important because it defines the ownership and management responsibilities over data and services (Figure 1). In hybrid deployments, layer ownership can be split between the customer and the provider and/or across multiple providers. Furthermore, this relationship may change over time; for example, a customer may handle the base load on an owned infrastructure, but burst to the public cloud to handle peak demand or system failures.

Due to the wide variety of deployment scenarios, the potential targets of cloud forensics can vary widely. Thus, the most productive approach for developing practical solutions is to start with specific (but common) cases and, over time, attempt to incorporate an expanding range. The

focus of this discussion is the forensics of cloud drive services, starting with the acquisition process.

2.2 Cloud Drive Forensics

The concept of a "cloud drive" is closely related to network filesystem shares and is almost indistinguishable from versions of the i-drive (Internet drive) that were popular the late 1990s. The main difference is that of scale – today, wide-area network (WAN) infrastructures have much higher bandwidth, which makes real-time file synchronization much more practical. Also, there are many more providers, most of which build their services in top of third-party IaaS offerings such as AWS.

Over the last few years, a number of forensic researchers have worked on cloud drives. Chung et al. [1] analyzed four cloud storage services (Amazon S3, Google Docs, Dropbox and Evernote) in search of traces left on client systems that could be used in criminal cases. They reported that the analyzed services may create different artifacts depending on specific features of the services and proposed a forensic investigative process for cloud storage services based on the collection and analysis of artifacts of cloud storage services recovered from client systems. The process involves gathering volatile data from a Mac or Windows system (if available) and then retrieving data from the Internet history, log files and directories. In the case of mobile devices, Android phones are rooted to collect data and iTunes is used to obtain information for iPhones (e.g., backup iTunes files). The objective was to check for traces of a cloud storage service in the collected data.

Hale [11] analyzed the Amazon Cloud Drive and discusses the digital artifacts left behind after an Amazon Cloud Drive account has been accessed or manipulated from a computer. Two methods may be used to manipulate an Amazon Cloud Drive Account: one is via the web application accessible using a web browser and the other is via a client application from Amazon that can be installed on the system. After analyzing the two methods, Hale found artifacts of the interface in the web browser history and cache files. Hale also found application artifacts in the Windows registry, application installation files in default locations and a SQLite database for tracking pending upload/download tasks.

Quick and Choo [16] discuss the artifacts left behind after a Dropbox account has been accessed or manipulated. Using hash analysis and keyword searches, they attempted to determine whether the client software provided by Dropbox had been used. This involved extracting the account username from browser history (Mozilla Firefox, Google Chrome and Microsoft Internet Explorer) and pursuing avenues such as

directory listings, prefetch files, link files, thumbnails, registry, browser history and memory captures. In a follow-up work, Quick and Choo [17] used a similar conceptual approach to analyze the client-side operation and artifacts of Google Drive and provide a useful starting point for investigators.

Martini and Choo [13] have researched the operation of ownCloud, a self-hosted file synchronization and sharing solution. As such, it occupies a slightly different niche because it is much more likely for the client and server sides to be under the control of the same person or organization. Martini and Choo were able to recover several artifacts, including sync and file management metadata (logging, database and configuration data), cached files describing the files the user stored on the client device and uploaded to the cloud environment or vice versa, and browser artifacts.

Outside of forensics, there has been some interest in analyzing the implementation of cloud drive services. An example is the work by Drago et al. [3, 4]. However, its focus was on performance and networking issues, and, although the results are interesting, their application to forensic practice is very limited.

2.3 Forensic Uses of Cloud Service APIs

Huber et al. [12] were among the first to utilize cloud service APIs as part of the forensic process. However, their main goal was to provide a context for an investigation by acquiring a snapshot of the social network of the investigative target via the Facebook Graph API.

With regard to commercial tools, Cloud Data eXplorer from ElcomSoft [6] offers the ability to acquire (via a service API) user artifacts from Google accounts, including profile information, messages, contacts and search history. However, no facilities are available to acquire cloud drive data, nor is there any support for services other than Google.

2.4 Summary

Previous work on cloud storage forensics has primarily focused on adapting the traditional application forensics approach to finding client-side artifacts. This involves blackbox differential analysis, where before and after images are created and compared to deduce the essential functions of the application. Clearly, the effectiveness of this approach depends on the comprehensiveness of the tests performed on a target system; ultimately, it is nearly impossible to enumerate all the eventualities that may have affected the state of an application. The process involves a labor-intensive reverse engineering effort, which requires sub-

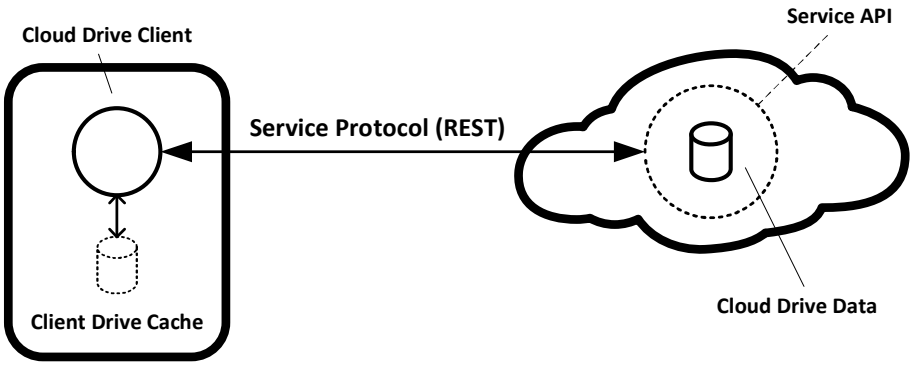


Figure 2. Cloud drive service architecture.

stantial human resources. Nevertheless, as discussed in the next section, the biggest limitation of client-side forensics is that it cannot guarantee the complete acquisition of cloud drive data.

3. Rationale for API-Based Acquisition

This section discusses the limitations of client-side acquisition and the benefits of API-based acquisition.

3.1 Limitations of Client-Side Acquisition

The fundamental limitation of client-side acquisition of cloud data is that it is an acquisition-by-proxy process. In other words, although it resembles traditional acquisition from physical media, the method does not target the authoritative source of the data, namely the cloud service. As illustrated in Figure 2, client content is properly viewed as a cached copy of cloud-hosted data. This simple fact has crucial implications for forensic acquisition.

Partial Replication. The most obvious problem is that there is no guarantee that any of the clients attached to an account have a complete copy of the cloud drive content. As a point of reference, Google Drive currently offers up to 30 TB of online storage (at a monthly cost of \$10/TB) whereas Amazon offers unlimited storage at \$60/year. As data accumulates, it will become impractical to maintain complete replicas of all devices. Indeed, based on current trends, it is likely that most users will not have a single device containing a complete copy of the data. From the forensic perspective, direct access is needed to cloud drive metadata to ascertain its contents. The alternative, blindly relying on

client cache, would result in an inherently incomplete acquisition with unknown gaps.

Revisions. Most drive services provide some form of revision history; the lookback period varies from 30 days to unlimited revision history depending on the service and subscription terms. This new source of valuable forensic information has few analogs in traditional forensic targets (e.g., Volume Shadow Copy service on Windows), but forensic investigators are not yet familiar with this evidentiary source. Revisions reside in the cloud and clients rarely have anything but the most recent versions in their caches. Thus, a client-side acquisition will miss prior revisions and will not even know that they are missing.

Cloud-Native Artifacts. Due to the wholesale movement to web-based applications, the digital forensics community must learn to handle a new problem – digital artifacts that do not have serialized representations in local filesystems. For example, Google Docs documents are stored locally as links to the documents that can only be edited via a web application. Acquiring an opaque link, by itself, is borderline useless – it is the content of the document that is of primary interest. It is often possible to obtain a usable snapshot of the web application artifact (e.g., in PDF), but this can only be accomplished by requesting it from the service directly; again, this cannot be accomplished by an acquisition-by-proxy process.

To summarize, the brief examination in this section reveals that the client-side approach to drive acquisition has major conceptual flaws that are beyond remediation. Clearly, what is needed is a different method that can obtain data directly from the cloud service.

3.2 Benefits of API-Based Acquisition

Fortunately, cloud services provide a front door – an API – to directly acquire cloud drive content. In broad terms, a cloud drive provides a storage service similar to that of a local filesystem; specifically, it enables the creation and organization of user files. Therefore, its API loosely resembles that of the filesystem API provided by the local operating system. Before the technical details of the proof-of-concept tool are described, it is necessary to make the case that the use of the API is forensically sound.

The main issue to address is that an API-based approach results in a logical – not physical– evidence acquisition. Traditionally, it has been an article of faith that obtaining data at the lowest possible level of abstraction results in the most reliable evidence. The main rationale is that

the logical view of the data may not be forensically complete because data marked as deleted is not shown. Also, a technically-sophisticated adversary may be able to hide data from the logical view. Until a few years ago, this view would have been reasonably justified.

However, it is important to periodically examine the accepted wisdom in order to account for new technological developments. It is outside the scope of this chapter to make a more general argument, but it should be noted that solid-state drives (SSDs) and even newer generations of high-capacity hard drives resemble autonomous storage computers rather than the limited peripherals of ten or more years ago. Some of them contain ARM processors and execute complex load-balancing and wear-leveling algorithms, which include background data relocation. Although they support, for example, block-level access, the results do not directly map to physical data layouts; this makes the acquired images logical, rather than physical. To obtain (and make sense of) a truly low-level representation of the data would increasingly require hardware blackbox reverse engineering. More than likely, this would lead to the wider acceptance of *de facto* logical acquisition as forensically sound.

In the case of cloud forensics, the case for adopting API-mediated acquisition is simple and unambiguous. According to Figure 2, the client component of the cloud drive (that manages the local cache) utilizes the exact same interface to perform its operations. Thus, the service API is the lowest available level of abstraction and is, therefore, appropriate for forensic processing. Furthermore, the metadata of individual files often include cryptographic hashes of their contents, which provide strong integrity guarantees during acquisition.

The service APIs (and the corresponding client software development kits for different languages) are officially supported by providers and have well-defined semantics and detailed documentation; this allows for a formal and precise approach to forensic tool development and testing. In contrast, blackbox reverse engineering can never achieve provable perfection. Similarly, acquisition completeness guarantees can only be achieved via an API – the client cache contains an unknown fraction of the content.

Finally, software development is almost always easier and cheaper than reverse engineering followed by software development. The core of the prototype tool described in this chapter is less than 1,600 lines of Python code (excluding the web-based GUI) for four services. An experienced developer could easily add a good-quality driver for a new (similar) service in a day or two, including test code. The code needs to be updated infrequently as providers strive to provide continuity and

Figure 3. Acquisition phases.

backward compatibility; any relevant additions to the API can easily be identified and adopted incrementally.

4. Tool Design and Implementation

Conceptually, acquisition involves three core phases: (i) content discovery; (ii) target selection; (iii) and target acquisition (Figure 3). During content discovery, the acquisition tool queries the target and obtains a list of artifacts (files) along with their metadata. In a baseline implementation, this can be reduced to enumerating all the available files; in an advanced implementation, the tool may leverage the search functionality provided by the API (e.g., Google Drive). During the selection process, the list of targeted artifacts can be filtered by automated means or by soliciting user input. The result is a (potentially prioritized) list of targets that is passed to the tool for acquisition.

Traditional approaches largely short-circuit this process by attempting to blindly acquire all the available data. However, this "acquire first, filter later" approach is not sustainable for cloud targets – the amount of data could be enormous and the available bandwidth could be up to two orders of magnitude less than the local storage.

The `kumodd` prototype described in this chapter is designed to be a minimalistic tool for research and experimentation that can also provide a basic practical solution for real cases. In fact, `kumodd` has been made as simple as possible to facilitate its integration with the existing toolset. Its basic function is to acquire a subset of the content of a cloud drive and place it in an appropriately-structured local filesystem tree.

4.1 Architecture

The `kumodd` tool is split into several modules and three logical layers: (i) dispatcher; (ii) drivers; and (iii) user interface (Figure 4). The dispatcher is the central component, which receives parsed user requests, relays them to the appropriate driver and returns the results. The drivers (one for each service) implement the provider-specific protocols via the respective web APIs. The tool provides two interfaces, a command-line interface (CLI) and a web-based GUI.

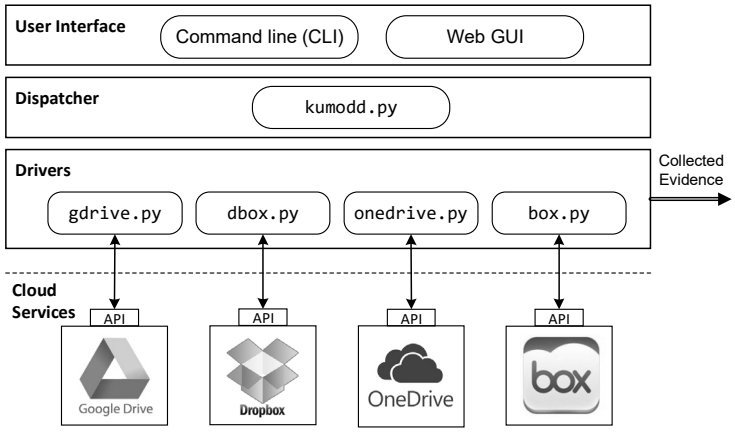


Figure 4. `kumodd` architecture.

4.2 Command-Line Interface

The general format of `kumodd` commands is:

```
python kumodd.py -s [service] [action] [filter]
```

The `[service]` parameter specifies the target service. Currently, the supported options are `gdrive`, `dropbox`, `onedrive` and `box` corresponding to Google Drive, Dropbox, Microsoft OneDrive and Box, respectively.

The `[action]` argument instructs `kumodd` on the action to be performed on the target drive:

- `-l` lists stored files as a plaintext table.
- `-d` downloads files subject to the `[filter]` specification.
- `-csv <file>` downloads the files specified by `<file>` in CSV format.

The `-p <path>` option is used to specify the path to which the files should be downloaded (and override the default, which is relative to the current working directory).

The `[filter]` parameter specifies the subset of files to be listed or downloaded based on file type: `all` (all files present); `doc` (Microsoft Office/Open Office document files: `.doc/.docx/.odf`), `xls` (spreadsheet files), `ppt` (PowerPoint presentation files); `text` files (text/source code); and `pdf` files.

In addition, some general groups of files can be specified: `officedocs` (all document, spreadsheet and PowerPoint presentation files); `image` (all image files); `audio` (all audio files); and `video` (all video files).

Some example commands are:

- List all the files stored in a Dropbox account:
  - `python kumodd.py -s dbox -l all`
- List the images stored in a Box account:
  - `python kumodd.py -s box -l image`
- Download the PDF files stored in a Microsoft OneDrive account to the Desktop folder:
  - `python kumodd.py -s onedrive -d all -l -p /home/user/Desktop/`
- Download the files listed in `gdrive_list.csv` from Google Drive:
  - `python kumodd.py -s gdrive -csv /home/user/Desktop/gdrive_list.csv`

User Authentication. All four services use the OAuth2 (`oauth.net/2`) protocol to authenticate a user and to authorize access to an account. When `kumodd` is used for the first time to connect to a cloud service, the respective driver initiates the authorization process, which requires the user to authenticate with the appropriate credentials (username/password). The tool provides the user with a URL that must be opened in a web browser, where the standard authentication interface for the service requests the relevant username and password.

The process for using Google Drive is as follows:

```
[title=Authentication Step 1: connect to \emph{Google Drive}]
kumo@ubuntu:~/kumodd$ python kumodd.py -s gdrive -d all
Your browser has been opened to visit:
https://accounts.Google.com/o/oauth2/auth?scope=
https%3A%2F%2Fwww.www.Googleapis.com...
...
```

Figure 5 shows the authentication steps: provide account credentials (left) and authorize application (right). After supplying the correct credentials and authorizing the application, the service returns an access code that the user must input in the command line to complete the authentication and authorization processes for the account. If the authentication is successful, the provided access token is cached persistently in

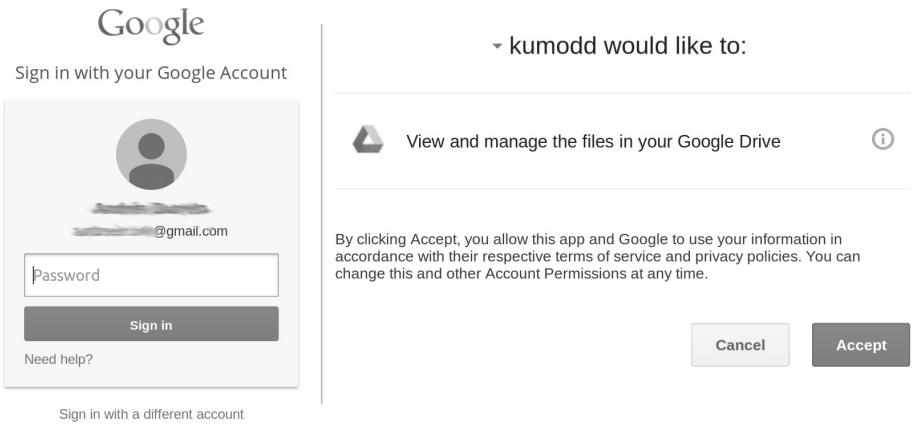


Figure 5. Authentication steps.

```
[caption=Sample processing output (with cached authorization token),
  label=lst:auth4]
kumo@ubuntu:~/kumodd$ python kumodd.py -s gdrive -d all
Working...
TIME(UTC) APPLICATION  USER FILE-ID REMOTE PATH REVISION LOCAL PATH
  HASH(MD5)
2015-06-25 03:48:43.600028 kumodd-1.0 example.dev@gmail.com
  1L-7o0rgPT2f6oX60OPtF4ZUFOmOJW1Crktr3DPril8o My Drive/ppt test
  v.2 downloaded/example.dev@gmail.com/My Drive/ppt test -
2015-06-25 03:48:44.951131 kumodd-1.0 example.dev@gmail.com
  1huaRTOudVnLe4SPMXhMRnNQ9Y_DUr69m4TEeD5dIWuA My Drive/revision doc
  test v.3 downloaded/example.dev@gmail.com/My Drive/revision doc
  test -
...
2015-06-25 03:48:54.254104 kumodd-1.0 example.dev@gmail.com
  0B4wSliHoVUbhUHdhZlF4NlR5c3M My Drive/test folder/stuff/more stuff/
  tree.py v.1 downloaded/example.dev@gmail.com/My Drive/test folder/
  stuff/more stuff/tree.py 61366435095ca0ca55e7192df66a0fe8
9 files downloaded and 0 updated from example.dev@gmail.com drive
Duration: 0:00:13.671442
```

Figure 6. Sample processing output (with cached authorization token).

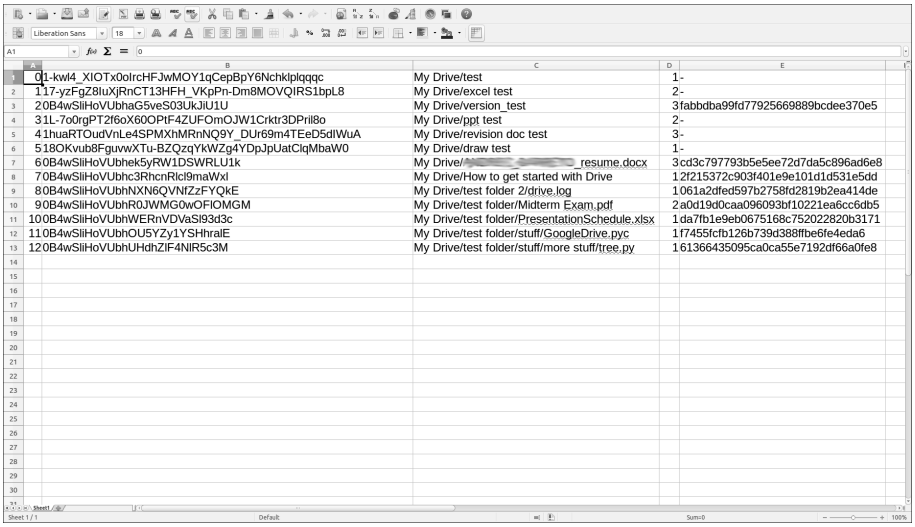
a `.dat` file saved in the `/config` folder with the name of the service. Future requests will find the token and will not prompt the user for credentials (Figure 6).

Content Discovery. In the current implementation, content discovery is implemented by the `list` (`-l`) command, which acquires the file

```
[caption=List of all files in a \emph{Google Drive} account
  (trimmed),label={lst:gdrive-ls}]
andres@ubuntu:~/kumodd$ python kumodd.py -s gdrive -l all
Working...
FILE-ID REMOTE PATH REVISION HASH(MD5)
...1qCepBpY6Nchklplqqqc My Drive/test 1 -
...oVUbhaG5veS03UkJiU1U My Drive/version_test 3 ...bcdee370e5
...
...oVUbhUHdhZlF4NlR5c3M My Drive/test folder/stuff/more stuff/
  tree.py ...2df66a0fe8
```

Figure 7. List of all files in a Google Drive account (trimmed).

metadata from the drive. As with most web services, the response is in the JSON format; the amount of attribute information varies based on the provider and can be quite substantial (e.g., Google Drive). Since it is impractical to show all the output, the `kumodd list` command provides an abbreviated version with the most essential information formatted as a plaintext table (Figure 7). The rest is logged as a CSV file in the `/localdata` folder with the name of the account and service.



	A	B	C	D	E
1	0	1-kwl4_XIOTx0oIrcHFJwMOY1qCepBpY6Nchklplqqqc	My Drive/test	1	-
2	1	17-yzFgZ8IuXjRnCT13HFH_VKpPn-Dm8MOVQlRS1bpL8	My Drive/excel test	2	-
3	2	0B4wSliHoVUbhaG5veS03UkJiU1U	My Drive/version_test	3	fabbdba99fd77925669889bcdee370e5
4	3	1L-7o0rgPT2f6oX60OPtF4ZUFOmOJW1Crktr3DPril8o	My Drive/ppt test	2	-
5	4	1huaRTOudVnLe4SPMXhMRnNQ9Y_DUr69m4TEeD5dIWuA	My Drive/revision doc test	3	-
6	5	18OKvub8FguvwXTu-BZQzqYkWZg4YDpJpUatClqMbaW0	My Drive/draw test	1	-
7	6	0B4wSliHoVUbhek5yRW1DSWRLU1k	My Drive/[redacted]_resume.docx	3	cd3c797793b5e5ee72d7da5c896ad6e8
8	7	0B4wSliHoVUbhc3RhcnRlcl9maWxl	My Drive/How to get started with Drive	1	2f215372c903f401e9e101d1d531e5dd
9	8	0B4wSliHoVUbhNXN6QVNfZzFYQkE	My Drive/test folder 2/drive.log	1	061a2dfed597b2758fd2819b2ea414de
10	9	0B4wSliHoVUbhR0JWMG0wOFlOMGM	My Drive/test folder/Midterm Exam.pdf	2	a0d19d0caa096093bf10221ea6cc6db5
11	10	0B4wSliHoVUbhWERnVDVaSl93d3c	My Drive/test folder/PresentationSchedule.xlsx	1	da7fb1e9eb0675168c752022820b3171
12	11	0B4wSliHoVUbhOU5YZy1YSHhralE	My Drive/test folder/stuff/GoogleDrive.pyc	1	f7455fcfb126b739d388ffbe6fe4eda6
13	12	0B4wSliHoVUbhUHdhZlF4NlR5c3M	My Drive/test folder/stuff/more stuff/tree.py	1	61366435095ca0ca55e7192df66a0fe8

Figure 8. Contents of the generated CSV file.

The stored output can be processed interactively using a spreadsheet program (Figure 8) or using Unix-style command line tools, thereby enabling a subsequent selective and/or prioritized acquisition.

Acquisition. As discussed above, the acquisition is performed by the download command (`-d`) and can be performed as a single discovery-and-acquisition step or it can be targeted by providing a list of files using the `-csv` option.

A list of downloaded files is displayed with information such as download date, application version, username, file ID, remote path, download path, revisions and cryptographic hashes. This information is also logged in the file `/downloaded/<username>/<service-name>.log`. Downloaded files are located in the `/downloaded/<username>/` directory. The complete original metadata files with detailed information about the downloaded files is stored in `/downloaded/<username>/metadata/` in the JSON format.

Revisions: The tool automatically enumerates and downloads all the revisions of the files selected for acquisition. The number of available revisions can be previewed as part of the file listing (Figure 8). During the download, the filenames of the individual revisions are generated by prepending the revision timestamp to the base filename. The filenames can be viewed using the regular file browser:

```
(2015-02-05T08:28:26.032Z) resume.docx    8.4kB
(2015-02-08T06:31:58.971Z) resume.docx    8.8kB
```

Arguably, other naming conventions are also possible, but the ultimate solution likely requires a user interface similar to the familiar file browser, but which also understands the concept of versioning and allows an analyst to trace the history of individual documents and obtain snapshots of a drive at particular points in time.

Cloud-Native Artifacts (Google Docs): A new challenge presented by the cloud is the emergence of cloud-native artifacts – data objects that have no serialized representation on local storage and, by extension, cannot be acquired by a proxy. Google Docs is the primary service considered in this work; however, the problem readily generalizes to many SaaS/web applications. A critical difference between native applications and web applications is that the code for the latter is dynamically downloaded at runtime and the persistent state of artifacts is stored back in the cloud. Thus, the serialized form of the data (usually in JSON) is an internal application protocol that is not readily rendered by a standalone application.

In the case of Google Docs, the local Google Drive cache contains only a link to the online location, which creates a problem for forensics. Fortunately, the API offers the option to produce a snapshot of the

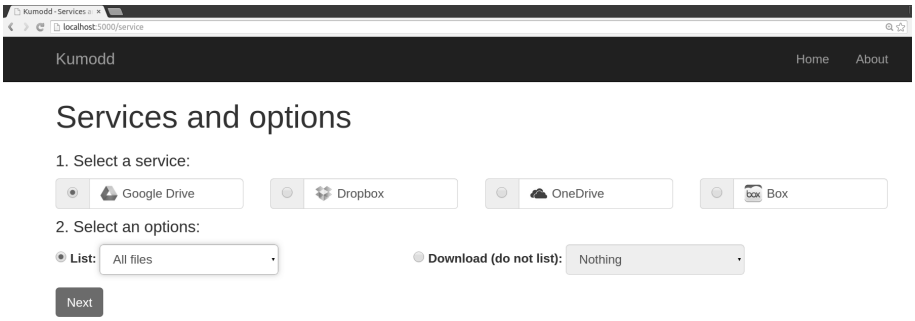


Figure 9. Web-based GUI: Service selection.

document/spreadsheet/presentation in several standard formats [10], including text, PDF and MS Office. At present, `kumodd` automatically downloads PDF snapshots of all Google Docs encountered during acquisition. Although this is a better solution than merely cloning the link from the cache, forensically-important information is lost because the internal artifact representation contains the complete editing history of the document. This problem is discussed later in the chapter.

4.3 Web-Based GUI

The `kumodd` tool provides an interactive web-based GUI that is designed to be served by a lightweight local web server. The GUI is started using the `kumodd-gui.py` module:

```
[title=Starting the web GUI]
python kumodd-gui.py
kumo@ubuntu:~/kumodd$ python kumodd-gui.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
```

After starting the server, the `kumodd` web-based GUI becomes available at `localhost:5000` and is accessible via a web browser. Note that, at this stage, the server should only be run locally; however, with some standard security measures, it could be made available remotely.

The web module employs the same drivers used with the command line application for authentication, discovery and acquisition. Its purpose is to simplify user interactions. For the simple case of wholesale data acquisition, the process can be accomplished in three button clicks.

After pressing the Get Started! button, the user is presented with the choice of the target service and the action to perform (Figure 9). After this step, a detail window presents a list of files and the option of choosing the files to be downloaded (Figure 10). After the files are

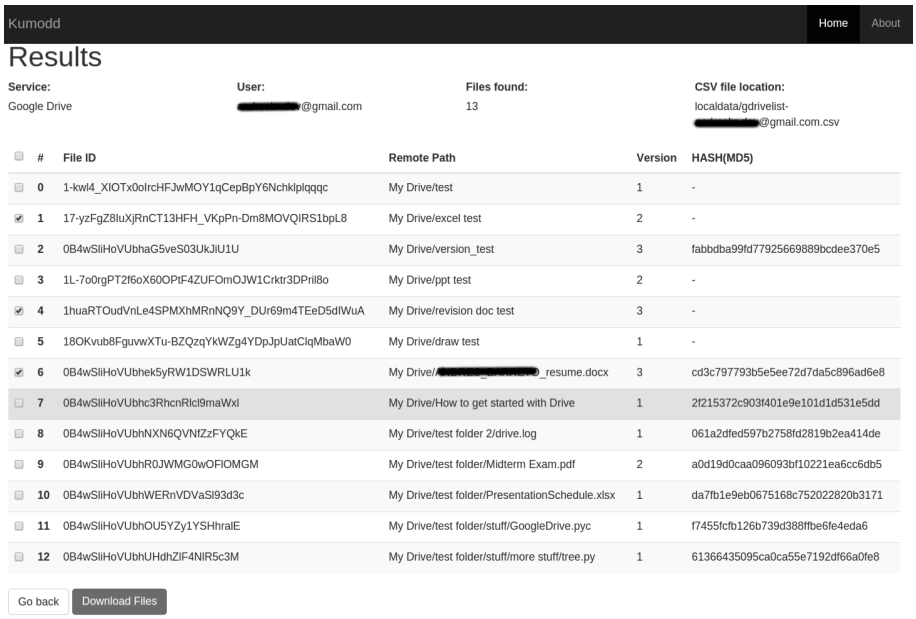


Figure 10. Web-based GUI: Target selection.

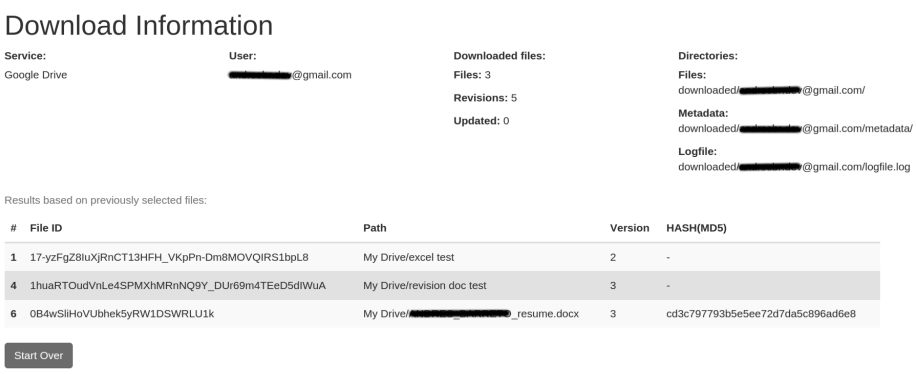


Figure 11. Web-based GUI: Acquisition results.

selected, a results screen presents the paths to the files and other relevant information (Figure 11). Note that every step of the process is also shown on the terminal.

4.4 Validation

To validate the tool, accounts were created with all the services and known seed files. Using the normal web interface for the respective

Table 1. `kumodd` download times and throughput.

Service	Average time (mm:ss)	Throughput (MB/s)
Google Drive	17:22	1.01
OneDrive	18:21	0.95
Dropbox	17:00	1.03
Box	18:22	0.95
Average	17:46	0.98

service, revisions were created and several Google Docs artifacts were created for Google Drive. A complete acquisition was performed using `kumodd` and the cryptographic hashes of the targets were compared. In all cases, successful acquisitions of all revisions were completed and snapshots were obtained of the cloud-only artifacts.

To obtain a sense of the acquisition rates, 1 GiB of data was split and stored in 1,024 files of 1 MiB each. The data was sourced from the Linux pseudorandom number generator to eliminate the potential influence of behind-the-scenes compression or de-duplication. The preliminary tests, which used the campus network, measured the throughput at different times of the day over a period of one week. No consistent correlations were observed.

To eliminate potential constraints stemming from the local-area network and Internet service provider infrastructure, the experiments were moved to Amazon EC2 instances (US-West, Northern California). For each of the four supported services, complete acquisition jobs were executed for seven days at 10:00 AM (PDT) in order to approximate daytime acquisition. Table 1 shows the average download times and throughput.

The Amazon EC2 results were entirely consistent with the on-campus results, which also averaged around 1 MB/s. The most likely explanation is that the bottleneck was caused by bandwidth throttling on the part of the cloud drive provider. It would appear that, at the free level, 1 MB/s is the implied level of service (at least for third-party applications).

5. Discussion

Several issues arose during the research and experimentation.

Integrity Assurance. An issue of concern is that not all services provide cryptographic hashes of file content as part of the metadata. In the experiments, this includes Dropbox (one of the largest providers), which only provides a "rev" attribute that is guaranteed to be unique

and is referred to as a "hash" in the API documentation [5]. However, the generating algorithm is unknown and the observed values are much too short (they fit in 64 bits) to be of cryptographic quality. However, it is known that, as of 2011, Dropbox was using SHA256 on 4 MiB blocks for backend de-duplication that, along with an incorrect implementation, led to the Dropship attack [2]. It is also known that the same provider uses cryptohashes to prevent the sharing of files for which it has received DMCA notices [15]. Therefore, providing a cryptohash for data object content would be a reasonable requirement for any cloud API used for forensic purposes.

Pre-Acquisition Content Filtering. One aspect not addressed in this work is the need to provide access to the search capability built into many of the more advanced services. This would allow triage and exploration with zero pre-processing overhead; some services also provide preview images that could be used for this purpose. Even with the current implementation, it is possible to filter data in/out by cryptohashes without the overhead of computing them.

Multi-Service Management and Forward Deployment. Cloud storage is a *de facto* commodity business with numerous providers vying for consumer attention. Thus, a number of cloud storage brokers have emerged to help users manage multiple accounts and optimize their services (often at the free level). For example, Otixo supports 35 different services and facilitates data movement among them. Forensic software needs similar capabilities to support cloud forensics.

In the near term, solutions that can be forward-deployed on a cloud provider infrastructure will be required. This will become necessary as cloud data grows much faster than the available bandwidth in a wide-area network, making full remote acquisitions impractical. The `kumodd` web interface is a sketch of such a solution; the forensic virtual machine instance could be co-located in the same data center as the target while the investigator controls it remotely. In this scenario, forensic analysis could begin immediately while the data acquisition could be performed in the background.

Long-Term Preservation of Cloud-Native Artifacts. It is mentioned above that Google Docs data objects are very different from most serialized artifacts that are familiar to analysts [19]. The most substantial difference is that the latter are snapshots of the states of artifacts whereas the former is literally a log of user edit actions since the creation of the document. This presents a dilemma for forensics. Should a

snapshot be acquired in a standard format such as PDF (like `kumodd`) and, thereby, lose all historical information? Or should the `changelog` be acquired and, thus, be dependent on the service provider (Google) to render it at any point in the future? Since neither option on its own is satisfactory, future research will attempt to develop a third option that will maintain the log and replay it independently of the service provider.

6. Conclusions

This research has three principal contributions. The first is a new acquisition model. It is clear that cloud drive acquisition cannot be performed on a client in a forensically-sound manner. This is because the client is not guaranteed to mirror all the data and has no facilities to represent file revisions and the content of cloud-native artifacts. The proper approach is to go directly to the source – the master copy maintained by the cloud service – and acquire the data via the API. In addition to being the only means for guaranteeing a forensically-complete copy of the target data, the API approach supports the reproducibility of results, rigorous tool testing (based on well-defined API semantics) and triage of the data (via hashes and/or search APIs). The overall development effort is significantly lower because the entire blackbox reverse engineering component of client-centric approaches is eliminated. As a reference, each of the four drivers for the individual services contains between 232 and 620 lines of Python code.

The second contribution is a new acquisition tool. The `kumodd` tool can perform cloud drive acquisition from four major providers: Google Drive, Microsoft OneDrive, Dropbox and Box. Although its primary purpose is to serve as a research platform, the hope is that it will quickly evolve into a reliable, open-source tool that will cover an expanding range of cloud services.

The third contribution is the elicitation of new research questions. This research has identified certain problems that must be addressed by the digital forensics community. In particular, it is necessary to develop a means for extracting, storing and replaying the history of cloud-native artifacts such as Google Docs. Additionally, a mechanism is needed to ensure the integrity of the data acquired from all providers. Moreover, it is necessary to build tools that can handle multi-service cases and operate in forward deployment scenarios.

Finally, it is hoped that this research will stimulate a new approach to cloud forensics and that it will serve as a cautionary note that simply extending client-side forensics holds little promise. Over the short term, this will mean expending extra effort to develop a new toolset. Over the

medium-to-long term, the emphasis on logical acquisition – which this work promotes – will help realize much greater levels of automation in the acquisition and processing of forensic targets.

References

[1] H. Chung, J. Park, S. Lee and C. Kang, Digital forensic investigation of cloud storage services, *Digital Investigation*, vol. 9(2), pp. 81–95, 2012.

[2] D. DeFelippi, Dropship (`github.com/driverdan/dropship`), 2016.

[3] I. Drago, E. Bocchi, M. Mellia, H. Slatman and A. Pras, Benchmarking personal cloud storage, *Proceedings of the ACM Internet Measurement Conference*, pp. 205–212, 2013.

[4] I. Drago, M. Mellia, M. Munafo, A. Sperotto, R. Sadre and A. Pras, Inside Dropbox: Understanding personal cloud storage services, *Proceedings of the ACM Internet Measurement Conference*, pp. 481–494, 2012.

[5] Dropbox, Core API Best Practices, San Francisco, California (`www.dropbox.com/developers/core/bestpractices`), 2016.

[6] ElcomSoft, ElcomSoft Cloud eXplorer, Moscow, Russia (`www.elcomsoft.com/ecx.html`), 2016.

[7] S. Garfinkel, A. Nelson and J. Young, A general strategy for differential forensic analysis, *Digital Investigation*, vol. 9(S), pp. S50–S59, 2012.

[8] Gartner, Gartner's 2014 hype cycle for emerging technologies maps the journey to digital business, Stamford, Connecticut (`www.gartner.com/newsroom/id/2819918`), August 11, 2014.

[9] Gartner, Gartner Hype Cycle, Stamford, Connecticut (`www.gartner.com/technology/research/methodologies/hype-cycle.jsp`), 2016.

[10] Google, Drive, Mountain View, California (`developers.google.com/drive`), 2016.

[11] J. Hale, Amazon Cloud Drive forensic analysis, *Digital Investigation*, vol. 10(3), pp. 295–265, 2013.

[12] M. Huber, M. Mulazzani, M. Leithner, S. Schrittwieser, G. Wondracek and E. Weippl, Social snapshots: Digital forensics for online social networks, *Proceedings of the Twenty-Seventh Annual Computer Security Applications Conference*, pp. 113–122, 2011.

[13] B. Martini and R. Choo, Cloud storage forensics: ownCloud as a case study, *Digital Investigation*, vol. 10(4), pp. 287–299, 2013.

[14] P. Mell and T. Grance, The NIST Definition of Cloud Computing, NIST Special Publication 800-145, National Institute of Standards and Technology, Gaithersburg, Maryland, 2011.

[15] K. Orland, Dropbox clarifies its policy on reviewing shared files for DMCA issues, *Ars Technica*, March 30, 2014.

[16] D. Quick and R. Choo, Dropbox analysis: Data remnants on user machines, *Digital Investigation*, vol. 10(1), pp. 3–18, 2013.

[17] D. Quick and R. Choo, Google Drive: Forensic analysis of data remnants, *Journal of Network and Computer Applications*, vol. 40, pp. 179–193, 2014.

[18] RightScale, RightScale 2015 State of the Cloud Report, Santa Barbara, California (`assets.rightscale.com/uploads/pdfs/RightScale-2015-State-of-the-Cloud-Report.pdf`), 2015.

[19] V. Roussev and S. McCulley, Forensic analysis of cloud-native artifacts, *Digital Investigation*, vol. 16(S), pp. S104–S113, 2016.