Google Drive Forensic Analysis via Application Programming Interface

A Thesis Presented to the Faculty of Jackson College of Graduate Studies

University of Central Oklahoma

In Partial Fulfillment of the Requirements of the Degree of

MASTER OF SCIENCE

in

FORENSIC SCIENCE

by

Shujian Yang

2015

Google Drive Forensic Analysis via Application Programming Interface

In Partial Fulfillment of the Requirements of the Degree of

MASTER OF SCIENCE

in

FORENSIC SCIENCE

by

Shujian Yang

A THESIS

APPROVED FOR THE W. ROGER WEBB FORENSIC SCIENCE INSTITUTE

2015

By [illegible] 12/2/15

Dr. Mark R. McCoy Committee Chairman Date

[illegible] 12/5/15

Dr. Wayne D. Lord Committee Member Date

[illegible] 12/2/15

Dr. Myung Ah Park Committee Member Date

Table of Contents

Chapter 1: Introduction 8
1.1 Background 8
1.2 Statement of the Problem 9
1.3 Purpose of the Research 11
1.4 Research Questions 12
1.5 Significance to the Field 12
1.6 Definitions 13
Chapter 2: Literature Review 14
2.1 Studies of Cloud Computing Forensics 14
2.2 Studies of Cloud Storages before Google Drive 15
2.3 Studies of Google Drive Forensics 16
Chapter 3: Research Method 19
3.1 Introduction 19
3.2 Application Development 20
3.2.1 Authentication 21
3.2.2 Data Integrity 23
3.2.3 Data Acquisition 24
3.2.4 Output 26
3.3 Experiment I 28
3.4 Experiment II 33
Chapter 4: Results and Analysis 37
4.1 Experiment I results 37

4.1.1 Basic file information 37
4.1.2 File user information 37
4.1.3 Deleted files recovery 38
4.1.4 File opening and editing via browser 39
4.1.5 Upload mechanism of Google Drive web application 39
4.1.6 Timestamps 40
4.1.7 Revisions 44
4.1.8 Summary 44
4.2 Experiment II results 45
4.2.1 vm2Aoff.001 45
4.2.2 vm2Aon.001 46
4.2.3 API application metadata record 47
4.2.4 Revisions 48
4.2.5 Summary 48
Chapter 5: Discussion 49
5.1 Research Problem 49
5.2 Answers to Research Questions 49
5.2.1 Answer toQ1 49
5.2.2 Answer to Q2 51
5.2.3 Other factors 52
5.3 Principles of applying API approach 53
5.3.1 Identify files by file ID 53
5.3.2 Separation of file content and metadata 54

5.3.3 Mind the volatility of evidence 55

5.4 Concerns about security 56

5.5 Limitations........ 57

5.6 Future works........ 58

5.7 Conclusion........ 59

References........ 60

Appendix........ 63

List of Tables

Table 3-1 Part of properties of Files resource........ 25

Table 3-2 APIs used in the testing application........ 26

Table 3-3 Experiment I file list 1 30

Table 3-4 Experiment I file list 2 30

Table 3-5 Experiment I file manipulation........ 31

Table 3-6 Sample result table........ 33

Table 3-7 Experiment II file list........ 34

Table 3-8 Experiment II file manipulation 1........ 35

Table 3-9 Experiment II file manipulation 2........ 35

Table 4-1 Basic information........ 37

Table 4-2 File user information........ 38

Table 4-3 Deleted files recovery 38

Table 4-4 Outcome of file opening and editing via browswer........ 39

Table 4-5 Outcome of re-uploading files 40

Table 4-6 Timestamps of files downloaded to local system 41

Table 4-7 Timestamps of uploaded only files 41

Table 4-8 Timestamps of viewed files 42

Table 4-9 Timestamps of downloaded files 42

Table 4-10 Timestamps of opened and edited file (direct modification) 42

Table 4-11 Timestamps of opened and edited file (create new entry) 43

Table 4-12 Timestamp of shared files 43

Table 4-13 vm2Aoff.001 analysis result 45

Table 4-14 vm2Aon.001 analysis result 46

Table 4-15 API application metadata record 47

List of Figures

Figure 3-1 Google sign in page 22

Figure 3-2 Authorization page 23

Figure 3-3 Part of a downloaded JSON format file opened in a text editor 28

Figure 5-1 Evidence Acquisition and Recording 55

Abstract

Rapid development of cloud computing brings challenges to digital forensic investigation, where traditional digital forensic tools and methodologies do not apply well. New approaches are needed to overcome emerged problems. This research focuses on analyzing a popular cloud storage service Google Drive in a forensically sound manner. The application programming interface (API) approach is chosen as the main method to perform digital forensic investigation. A sample application is developed to acquire evidence from Google Drive. Experiments were then conducted to evaluate its effect based on results. By comparing the results with other approaches, the API approach proves to be effective and reliable for digital forensic examiners and forensic software developers to consider as available tool in their arsenal.

Chapter 1: Introduction

1.1 Background

The information technology (IT) industry is experiencing huge changes in recent years. The concept of cloud computing rapidly spread over the IT industry and was successfully implemented. Thanks to the promotion of large companies, such as Google and Microsoft, and the devotion from open source communities to projects like Apache Hadoop, more people are enjoying the convenience and benefits given by cloud computing nowadays. On one hand, enterprise users lower their cost significantly by switching from old styled direct investment in expensive infrastructure to cheap scalable business services. On the other hand, individual users improve their life on the internet by using cheap or even free services which they had never experienced before. These are often services offered by public cloud service providers as the outcome of cloud computing development.

Cloud storage is one example of these services. It allows users to store files in a cloud service provider's infrastructure and access them anywhere anytime as long as the internet connection is established. These services are usually relatively cheap compared to traditional storage media or even free, which makes them welcomed by individual users. The demand for cloud storage services has been increasing since recent years. In 2012, a market intelligence firm iHS iSuppli reported that the number of cloud storage service users could be 375 million and could reach 700 million by the end of 2014 (Lardinois, 2012). In 2014, research and consulting firm Markets and Markets predicted that the cloud storage market could worth $56.57 billion in 2019 (ITPP in association with HP, 2014). By contrast, the market of hard drive was reported to drop due to the competition from cloud storage service (Merriman, 2014). Such a huge market of cloud storage encourages many companies to offer their cloud storage service to the public individual users. At the time of writing, major competitors in this area include Dropbox, Google Drive and Microsoft OneDrive (Griffith, 2014).

Google Drive is the cloud storage service introduced by Google in 2012. As of March 2015, according to official documents on Google's website, Google Drive offers 15 Gigabytes of free space to regular users by default, where it can be upgraded to 100 Gigabytes or more by charging as low as $1.99 per month. The file system of Google Drive allows storage of any file types while offering features such as sharing, Gmail attachment and old versions. Users of Google Drive can access their files by using supported web browser to log in to Google's website. Besides that, Google also offers official client software which can run on PC, Mac, Android or iOS devices, allowing users to enjoy their services in an alternative way. According to a Google announcement, the number of active users had reached 190 million in June, 2014 (Google, 2014).

It is worth noting that Google released an ungraded paid version of Google Drive, Google Drive for Work in June 2014, targeting business users. Users of Google Drive for Work obtain larger space, more security and more control over accounts. These tools are intended to help business teams increase their productivity (Johnston, 2014). Another similar product is Google Drive for Education, which targets school students and faculty members. Both of the two upgraded versions of Google Drive include all features of the regular version.

1.2 Statement of the Problem

The growing usage of cloud storage service increases the chance of cloud storage being involved in illegal activities. On one hand, many individual users store their personal data in cloud storage. Sometimes they are even totally unaware of that due to the automatic upload function of their devices. The personal data stored in the cloud storage may be vulnerable to attacks from unauthorized malicious hackers. One of the most well-known incidents recently is the leaking of multiple celebrity personal photos stored in iCloud, Apple Company’s cloud storage service in 2014 (Duke, 2014). On the other hand, criminals may store incriminating records in the cloud storage. This includes scenarios like drug dealers storing drug transaction records in the cloud storage, or a person sharing child pornography to others via a cloud storage account.

Such issues raise some concerns from the digital forensics community. In the past decades, digital forensic examiners have been working on many kinds of traditional storage media, such as hard drives and optical discs, and developed standard procedures to collect evidence within effectively. However, there are many new features of cloud storage compared with traditional storage. Some of these features bring new challenges to digital forensic examiners while some others may assist them in ways never imagined before.

Among these features, the one that hinders traditional digital forensic methodology the most must be the unreachability of infrastructure, that is, the hardware. The traditional acquisition stage of the digital evidence process requires full access to the target storage hardware and file system, which is impractical in the case of cloud storage service. Meanwhile, the verification stage is also troubled by cloud storage. The forensic examination process requires the integrity of evidence to be maintained as much as possible. Since examiners do not have control over cloud storage service infrastructure, therefore classic integrity preservation measures, for example, write protection by changing BIOS setting, do not apply to cloud storage. Even if examiners have seized the device used to connect to cloud storage, someone can still connect to the same cloud storage and change data in it. Thus, the volatility of data stored in cloud storage requires more attention from examiners.

A certain amount of related research has been performed by digital forensic practitioners to overcome these challenges. With their efforts, different approaches are developed. Marturana, Me, and Tacconi (2012) studied evidence left by Dropbox. Cookies, internet history and temporary files related were found in the system. Quick and Choo (2014) did research aimed to recover as much evidence as possible from cloud storage services, including Dropbox, Microsoft SkyDrive (now named OneDrive) and Google Drive. Their research revealed lots of mechanisms unnoticed before, for example, the stored login password in IE and registry remnant in Windows system. Federici (2014) introduced a whole new different measure to collect information. In his research, a library based on Google Drive API was written to collect evidence in Google Drive along with some other cloud storage services. This approach revealed more evidence left in cloud storage than previous methods. Nevertheless, there are still some challenges, leaving gaps for further research to be filled.

Despite the existing challenges, cloud storage also brings opportunities for digital forensic examiners to discover additional valuable data in relation to traditional storage media and file systems. In Federici's research, the old versions of a modified file stored in cloud storage can be retrieved (Federici, 2014). However, there are still many hidden features, such as file sharing, which have not yet been studied thoroughly yet.

In summary, in order to perform effective forensic examination over cloud storage, researchers have developed several approaches. Among them, the API approach has better coverage than the others to meet examiner's need. However, the validity of the API approach still needs to be further tested, while some potential of this approach has not been discovered yet.

1.3 Purpose of the Research

The purpose of this research was to fully utilize the Google Drive API approach in a forensically sound manner to collect as much valuable evidence as possible in a Google Drive account while maintaining integrity of evidence. Although there are already some approaches to acquire evidence in Google Drive, digital forensic examiners need more reliable and convenient methods to assist them in the tasks related to cloud storage. This study was expected to prove the effectiveness of the API approach being used to collect evidence in Google Drive by analyzing test files via API and compare the result with actual values.

1.4 Research Questions

There are two research questions to be answered:

Q1: What types of evidence in Google Drive can be found via the API approach?

Q2: How was the integrity of evidence maintained during the acquisition via the API approach?

1.5 Significance to the Field

Traditional digital forensics focus on digital evidence left in accessible media such as hard drives. With the development of cloud storage, digital forensic examiners need to follow the trend. However, digital forensic examiners still lack the adequate tools to handle cloud storage problems. National Institute of Standards and Technology (NIST) maintains a list called Computer Forensics Tool Catalog, which gathers information of lots of available digital forensic tools divided into many categories. Unlike other categories which list quite an amount of tools, the “cloud services” category contains only one tool named Internet Evidence Finder (IEF) (National Institute of Standards and Technology, 2014).

The goal of this research is to provide some guidelines and recommendations for further cloud storage oriented forensic tool development. If succeeded, this research can make a significant contribution to digital forensic communities to handle possible Google Drive involved criminal scenarios in the future. Digital forensic examiners could find more traces left to provide vital evidence accepted by court to support trial.

1.6 Definitions

API: Application Programming Interface. Interface used to gain access to a program or service (Sommerville, 2011)

Drive API: The API released by Google to access Google Drive service.

Cloud Storage: A service offered by cloud service providers, allowing users store files on cloud service providers’ infrastructure.

Google Drive: A cloud storage service offered by Google since 2012.

REST: Representational State Transfer. It is a design philosophy of network-based software. A RESTful API enables a server offers it service in a stateless manner using uniform interface (Fielding, 2000).

Chapter 2: Literature Review

2.1 Studies of Cloud Computing Forensics

Since Google Drive was introduced to the public in 2012, digital forensic examiners have tried several methods to solve the problem of collecting evidence from it. However, some researches about cloud computing, the cornerstone of Google Drive, existed before Google Drive was born. The literature review will address research related to Google Drive forensic examinations.

Cloud storage, such as Google Drive, is one of the products of the development of cloud computing. Research about digital forensics in the world of cloud computing began before cloud storage service was well known by the public. In 2010, Taylor, Haggerty, Gresty, and Hegarty identified several challenges most likely to be met during cloud computing forensic examination. Although cloud storage service is not explicitly mentioned, they did talk about "digital evidence resides within a public cloud," which is similar to the concept of cloud storage service. They found that cloud computing can be helpful for digital forensic investigation because data was backed up in the cloud. However, evidence stored within the cloud is "more ethereal and dynamic." Some data was not stored in the traditional operating system but a virtual environment, where it will be lost when the user exists. Some data of a file concerned by forensic examiners, such as created date and modified dates, may get lost in the cloud. The authors concluded due to the difference between cloud computing and traditional systems, "it may potentially be difficult to obtain digital evidence to the same standard as that currently obtained from traditional server-based systems" (Taylor, Haggerty, Gresty, & Hegarty, 2010).

Taylor, Haggerty, Gresty, and Lamb (2011) later identified more challenges of digital forensics in a cloud computing environment. Physical seizure, which is a standard operation conducted by digital forensic examiners to preserve evidence, is difficult. Data in cloud computing is stored in distributed systems, or even large data centers, making the law enforcement agency unlikely to seize all related computing assets. Another problem to be solved is encryption. To ensure the security of end users' data, cloud computing providers tend to implement some kinds of security mechanisms, including data encryption and user authentication. Although traditional techniques such as password keyword searching still work in some cases, beating encryption still creates a burden on examiners.

Article of Taylor, Haggerty, Gresty, Hegarty (2010) and Taylor, Haggerty, Gresty, Lamb (2011) revealed main problems encountered by digital forensic examiners in a cloud computing environment, but they did not provide specific solutions. More research was conducted by others to overcome these problems.

2.2 Studies of Cloud Storages before Google Drive

Before Google Drive, some other cloud-based services already existed in the market. Marturana, Me, and Tacconi (2012) studied the problem of acquiring digital evidence in such services. One of the studied targets was Dropbox, a popular cloud storage service very similar to Google Drive later. They used network sniffer software to identified network protocols used during cloud storage service communication and later performed traditional digital forensic examinations on a hard disk to discover traces left by web browsers and client software. Cookies, internet history and temporary files related were found in the system.

Chung, Park, Lee, and Kang (2012) did other research, targeting more cloud storage services, including Amazon S3, Dropbox, Evernote and Google Docs. Unlike the research conducted by Marturana et al. (2012), where only the Windows platform is considered, Chung et al. performed tests on Windows, Mac, iPhone and Android phone. Their study focused on artifacts left in these platforms by cloud storage client software and web browsers. The result varied from service to service. In summary, discovered artifacts include internet temporary files created by web browsers and application files stored in the operating system to support the services. These artifacts could be used to indicate the usage of cloud storage services. Other obtained information included login usernames, time of cloud storage operations and snapshots of transmitted files.

2.3 Studies of Google Drive Forensics

After Google Drive was released in 2012, Quick and Choo (2014) conducted research aimed to recover as much evidence as possible from Google Drive. In this research, several virtual machines were created with different web browsers and Google Drive client software installed. The research then performed different tasks on the Google Drive account, such as browsing files and downloading. After that, images of virtual hard drives of these virtual machines were made and examined with traditional digital forensic tools, including AccessData FTK and Guidance software EnCase.

Quick and Choo revealed many remnants left by Google Drive. They used web browser to access history files left by Google Drive, which can be found by searching keywords like “drive.google.com”. Username of a Google Drive account could possibly be found in database files written by web browsers. If Google official client software was used, SQLite database files created by client software were discovered in system application folders. By opening these files, useful information such as username, uploaded filenames, created time could be recovered. The research also introduced a method to access Google Drive account without knowing the username and password if client software was installed, that is, run the image in a virtual environment and allow client software to synchronize with registered account automatically.

Data integrity is another aspect digital forensic examiners care about. Digital forensic investigation demands content of evidence unchanged as when they were collected. In addition, being part of evidence, the metadata of an acquired file is as important as the file content. In traditional digital forensics, the preservation of data integrity is usually guaranteed by making forensic image of evidence media and manipulating only the image instead of the evidence itself. However, in the case of cloud storage, such measure does not apply due to the inaccessibility of storage hardware. So is it still possible to preserve data integrity when acquiring evidence from cloud storage such as Google Drive with available measures? To answer the question, Quick and Choo (2013) designed an experiment. They uploaded some files to a Google Drive account, then later downloaded the files and examined metadata via web browser and client software measures, as their another research illustrated (Quick & Choo, 2014), respectively. Quick and Choo calculated and compared hash values of downloaded files and original files, finding they were all matched. This proved that file content was not modified during acquisition via either measure. However, the timestamp of files were changed. Neither web browser nor client software measures could preserve the original file timestamps successfully.

Quick and Choo’s research (2014) used only existing tools offered by Google, i.e., web applications via browsers and client software, as core methods to collect data from cloud storage as normal individual users do. Federici (2014) introduced a whole new different measure to collect information. In his research, Federici developed an application called cloud data imager used to collect evidence from Dropbox, Microsoft SkyDrive (now renamed as OneDrive) and Google Drive. The application was designed to connect to three cloud storage services. Then it generated a unified interface to allow visits to the services, provided that valid usernames and passwords were given. Through the unified interface, some common actions can be performed such as listing files and folders, listing file revisions and download files.

The most important benefit of Federici's work was it bypassed normal software not designed for forensic analysis purpose, leading to far less metadata lost or alternation. Some special features of Google Drive, such as previous versions, are also covered.

However, there were still limitations in the study. In Federici's study, Dropbox, Microsoft SkyDrive and Google Drive were treated equally in order to generate the unified interface. The generated unified interface did offer convenience to examiners, but in the price of leaving some unique features of each service not processed. Also, the integrity of evidence collected via the API approach is not fully verified. For example, a file in Google Drive was modified. The application could tell when it was modified. But by whom it was modified? How it was modified? These are questions to be answered.

Google Drive has been evolving since it was introduced in 2012. More functions were added to provide richer user experience. Today, users can perform more actions on stored files than they could before. However, in the previous studies, researches spent most of time on discussing the evidence gathered from cloud storage but very few time on user actions. This is not very good since evidence is the result while user action is the cause. A full evaluation of Google Drive forensic analysis should consider both results and causes. Only by this can we better infer what actually happened based on collected evidence. There are still works to do to discover the full potential of Google Drive in forensic perspective.

Chapter 3: Research Method

3.1 Introduction

This research is designed to study Google Drive, a very popular cloud storage service, from the standpoint of digital forensics. In the traditional digital forensics discipline, evidence to be collect resides in reachable hardware devices, for example, hard drive, flash drive, cell phone memory. Many outstanding hardware and software products are developed to retrieve these kinds of evidence effectively. However, in cloud storage services, the target evidence is mainly stored remotely in cloud service providers' infrastructures, which is very unlikely to be handled by traditional forensic tools.

To fill the gap of traditional forensic tools, some researchers employed software developed by cloud storage providers to perform digital forensic investigation. In Quick and Choo's research, web application and installed client software, both developed by Google official, were used to collect evidence from Google Drive (Quick & Choo, 2014). Although plenty of valuable information was obtained, these twos approaches have a flaw: they are designed for normal users, not digital forensic examiners. Many types of metadata of files concerned by digital forensic examiners could not be obtained in this way. Moreover, the tampered metadata and lack of necessary evidence validation process makes it not a preferred measure for digital forensic examiners when considering the possibility of collected evidence rejected in court. To overcome the problem, the application programming interface (API) approach is discovered.

API refers to "an interface, generally specified as a set of operations, that allows access to an application program's functionality" (Sommerville, 2011, p. 734). Cloud service providers offer basic services to public users, including individuals and business users. They provide some software for normal individual users to use their services. However, this kind of software can not meet the unique requirements from business users, or people with special needs. Thus, the API is introduced by cloud service providers. API provides interface to allow applications to directly access the service. The API released by Google to access Google Drive service is called Drive API.

In this research, the latest version of Drive API v2 from Google, updated in February 2015, was utilized to discover the possibility of it being used as a tool for digital forensic examiners to collect evidence in Google Drive accounts and answer the research problems. A sample application was written to demonstrate how the Drive API works.

There were two stages of this research: application development stage and experiment stage.

3.2 Application Development

Application development stage focused on developing a sample application that employed Google Drive API to communicate with a Google Drive account. The platform and programming language used were .NET 4.5 and C# 5.0. The basic requirements of the sample application were:

1. Authentication - Be able to pass the authentication process to get access to designated Google Drive account (target account).
2. Data Integrity - Make minimal changes to evidence as possible in target account.
3. Data Acquisition - Be able to retrieve maximum forensically valuable information from the target account. The preferred types of information may include but not limit to: file data, file metadata, file revisions, account information, user or visitor information.
4. Output - Be able to export obtained information in acceptable manners.

The Drive API which Google offers to developers is RESTful. The term “REST” refers to a set of constraints for designing network-based software architecture. A RESTful API enables a server to offer its service in a stateless manner using uniform interface (Fielding, 2000). Regarding Google Drive API, an application can request resources from Google Drive using standard HTTP methods and receive data in a standard format as a response. The URIs for which these requests destined are mostly relative to https://www.googleapis.com/drive/v2. This process is not restricted to certain programming languages.

Nevertheless, in order to make software development easier, Google offers software development kits (SDK). SDK contains client libraries written in popular programming languages and detailed documentations of Drive API. With assistant of these tools, programmers can develop applications faster in the languages they are familiar with.

The following section will discuss how to use Drive API, .NET client library from Google and other resources to develop application which meets the four requirements listed at the beginning of this section.

3.2.1 Authentication

Google Drive uses OAuth 2.0 for authentication. OAuth 2.0 is a protocol established by the Internet Engineering Task Force (IETF), a non-profit organization who provides internet standards. It provides third-party applications the ability to use HTTP services, for example, the Google Drive (Internet Engineering Task Force, 2012).

By applying OAuth 2.0 protocol, the application first receives client credential, which contains two strings called client id and client secret respectively, from Google. With the credential, the application connects to the Google Authorization Server. The Google Authorization Server will authenticate the user by checking username and password. If succeeded, the server then sends an access token to the application, which allows the application to access the Google Drive service (Google, 2015d). During this process, the application itself does not process username and password from user, which strengthens security.

In .NET client library, Google provided a method called AuthorizeAsync in class GoogleWebAuthorizationBroker to help installed applications running on Windows authenticate the user. When this method is called, the default browser will be opened. If the user is not signed in with a Google account in this browser session, a sign in page appears, asking the user to input email (as username) and password (Figure 3-1).

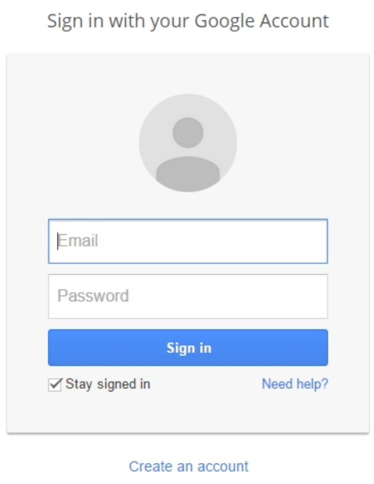
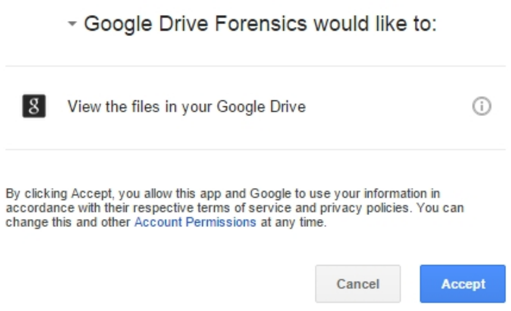


Figure 3-1 Google sign in page

If the user successfully signs in, an authorization page is displayed (Figure 3-2). The page shows the name of application and what it is authorized to do over the Google Drive account. Once the user clicks the “Accept” button, the authentication process will be completed, allowing the application to execute the rest of the codes and do the jobs.



Google Drive Forensics would like to:

View the files in your Google Drive

By clicking Accept, you allow this app and Google to use your information in accordance with their respective terms of service and privacy policies. You can change this and other Account Permissions at any time.

Figure 3-2 Authorization page

In the process above, the user of the application must enter email and password in order to gain access to the target Google Drive account. However, this step is possible to bypass. Once the OAuth 2.0 process begins and opens the page, if the user has already signed in with a Google account with the same browser, or cookies are used to keep the user stay signed in, then the page in Figure 3-1 will not appear. Instead, the user will see Figure 3-2 directly, skipping the stage where username and password are needed. This feature can be extremely helpful to digital forensic examiners when username or password of the target Google account is unknown.

3.2.2 Data Integrity

Data integrity is critical to digital forensic investigation. During the evidence collection, the chance of evidence being altered should be as low as possible. In Google Drive API, the level of data alternation can be controlled by designating scope.

Scope is part of the Google Drive API OAuth 2.0 protocol. Every application connected to a Google Drive account must be assigned with one or more scopes, which restrict the access of the application. For example, a drive scope permits full access of all files, while a drive.install scope allows user install an app. The scope of an application can be examined by the user in the authorization page as Figure 3-2 shows in the middle part.

To reduce the chance of possible data alternation, the most suitable scope to be used drive.readonly. By applying drive.readonly scope only, the forensic application can read file metadata and file content, but is not allowed to alter them. This approach can significantly increase the chance of preserving data integrity. Figure 3-2 illustrated the authorization page when only drive.readonly scope is applied.

Nevertheless, this cope does not necessarily guarantee full data integrity. The question of whether data integrity can be fully achieved can be answered only via experiment, which will be discussed in the experiment stage.

3.2.3 Data Acquisition

With the help of RESTful API, an application can request different types of resources from Google Drive by different methods. These methods are implemented with standard HTTP requests.

To get started, listing all files stored in the account will be useful. This can be done by applying the “list” method, which is capable of listing all files in the account, including those in subfolders or was trashed. The list method is compatible with drive.readonly scope, meaning it only reads to the files in the Google Drive account without writing to it and meets the data integrity requirement. The implementation of list method also confirms this because it uses only HTTP GET request, ending in receiving Files resources, which will be introduced later, from the server without modifications.

Among many resources offered by Google Drive, “Files” is the important one. The Files resource is the collection of information about files stored in a Google Drive account. By requesting Files via API, an application can obtain properties of one or more files. A complete list of file properties can be found on Google Developers website (Google, 2015c). Important properties with forensic value are listed in Table 3-1:

Table 3-1 Part of properties of Files resource

Property	**Description**
id	A unique ID for each file.
title	Title of a file.
originalFilename	Original filename.
fileExtension	File extension.
md5Checksum	MD5 hash value.
fileSize	Size of file in bytes.
mimeType	MIME type of a file.
createdDate	Create time.
modifiedDate	Last modified time.
modifiedByMeDate	Last time of modified by current user.
owners[]	List of owners of this file.
lastModifyingUser	Last user who modified this file.
lastViewedByMeDate	Last time this file was view by current user.
downloadUrl	Download URL for this file

Note the term “file” in Drive API actually refers to an entry of its file system. An entry can be not only an actual file but also other form of data such as a folder or an entry created by apps like Google Docs. Based on the type of the file, some properties may or may not exist. For example, only actual file has fileSize property. A folder would have only a title but no original filename. If a file is uploaded to Google Drive, its title and original filename will share a same value, contrary to entry created by Google Docs, where titles and original filenames may have different values, or even null values.

For the purpose of clarity, in the rest of this article, the term “**entry**” is used to refer to a record in Google Drive’s file system, while “**file**” refers to entry that has actual content, has file size and can be downloaded directly. Whether an entry is an actual file can be determined by the value of fileSize property of the entry. If fileSize is a non-null value, it is an actual file. Otherwise, it is an entry that could not be downloaded directly

Properties of an entry provide metadata. If an entry is a file, its content can be acquired by sending HTTP GET request using the entry’s id property. In Google Drive, only actual files can be downloaded directly in this way. For those entries that are not files, extra measures can be used to download their contents. For instance, a Google Docs entry does not have fileSize property. However, its properties include download links, allowing users to export its content in different formats such as Microsoft Word document or PDF.

As one of the most unique features of cloud storage, file revision history can also be visited by requesting Revisions resource. According to Google's advertisement, revisions of most file types would be kept for as far as 30 days (Google, n.d.). Once a file stored in Google Drive is modified, Google Drive generates a revision record, in which the content and metadata of this version is written. Google Drive API provides rich support to revision information. Items listed in a revision record include file size, md5 checksum, modified time, last modifying user and so on. To request Revisions resource, the application must send the HTTP GET request along with the file id and corresponding revision id.

In summary, the APIs used in the testing application are listed in the following table. All URIs are relative to https://www.googleapis.com/drive/v2.

Table 3-2 APIs used in the testing application

HTTP request	**Purpose**
GET …/files	List all entries in a Google Drive account.
GET …/files/fileId	Get a certain entry's data.
GET …/files/fileId/revisions/revisionId	Get a certain entry's revisions.

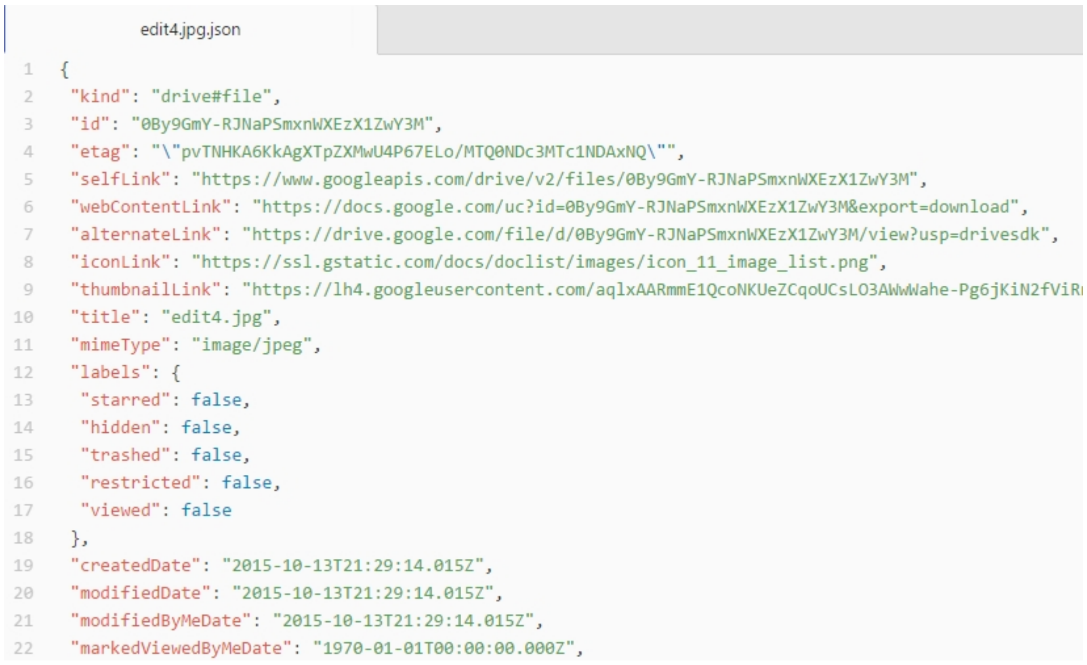
3.2.4 Output

There are two types of data to be exported. The first is file content, which can be easily saved by creating new files using response stream from download requests. The second type is entry metadata. It is important to keep a record of all entry metadata independently instead of relying on metadata of downloaded files. There are three reasons to do so:

1. Some entries can not be downloaded, as explained above.
2. The download process may change the metadata, resulting in different metadata as they were in Google Drive.
3. Some of the metadata provided by Drive API do not exist in local file system, meaning they will be lost when creating downloaded files.

Thus, entry metadata must be exported in some kind of format and saved as a log. Popular formats to be considered include Comma Separated Values (CSV) or JavaScript Object Notation (JSON).

In fact, when requesting metadata of an entry using RESTful API, the response from server is already a JSON formatted data stream containing all metadata. Thus one simple practice to maintain a metadata record is simply storing all responses from the server as JSON files. Each entry corresponds to one JSON file in which all its metadata is stored. The stored metadata can be read and analyzed conveniently due to popular support of reading JSON file provided by many programming languages.



```
{
 "kind": "drive#file",
 "id": "0By9GmY-RJNaPSmxnWXEzX1ZwY3M",
 "etag": "\"pvTNHKA6KkAgXTpZXMwU4P67ELo/MTQ0NDc3MTc1NDAxNQ\"",
 "selfLink": "https://www.googleapis.com/drive/v2/files/0By9GmY-RJNaPSmxnWXEzX1ZwY3M",
 "webContentLink": "https://docs.google.com/uc?id=0By9GmY-RJNaPSmxnWXEzX1ZwY3M&export=download",
 "alternateLink": "https://drive.google.com/file/d/0By9GmY-RJNaPSmxnWXEzX1ZwY3M/view?usp=drivesdk",
 "iconLink": "https://ssl.gstatic.com/docs/doclist/images/icon_11_image_list.png",
 "thumbnailLink": "https://lh4.googleusercontent.com/aqlxAARmmE1QcoNKUeZCqoUCsLO3AWwWahe-Pg6jKiN2fViR
 "title": "edit4.jpg",
 "mimeType": "image/jpeg",
 "labels": {
  "starred": false,
  "hidden": false,
  "trashed": false,
  "restricted": false,
  "viewed": false
 },
 "createdDate": "2015-10-13T21:29:14.015Z",
 "modifiedDate": "2015-10-13T21:29:14.015Z",
 "modifiedByMeDate": "2015-10-13T21:29:14.015Z",
 "markedViewedByMeDate": "1970-01-01T00:00:00.000Z",
```

Figure 3-3 Part of a downloaded JSON format file opened in a text editor

The experiment stage followed after the program development stage. In this stage, two sets of experiments were performed in order to compare API application approach with web browser and client software approaches respectively. The reason to use two sets of experiments instead of one was that web browser and client software approaches requires different experimental environments.

In the experiment stage, files from Govdocs1 distributed by Digital Corpora were used as test subjects. Govdocs1 is a library of nearly one million files, which can be used freely by digital forensic examiners for research (Digital Corpora, 2014).

3.3 Experiment Ⅰ

Experiment I was designed to compare API approach in relation to the browser approach. In this experiment, Google Chrome browser was used to upload files and then perform multiple operations. During this process, the browser was employed as a tool to visit the web interface, namely the web pages, designed by Google engineers. All information gathered through browser was limited to the one given by Google Drive web interface. Thus, in the following discussion, web interface was considered as the source of information while the browser was just a measure to collect the information. The goal of this experiment was to find the outcomes of these operations, collect evidence, and evaluate whether evidence collected by different approaches was good to be used to infer what actually happened.

The environment was prepared as follows. Two different virtual machines named vm1A and vm1B were set up with the same version of operating system and Google Chrome web browser installed. Two newly opened free Google Drive accounts acnt1A and acnt1B were set up with no existing contents modified as target accounts. The first step was preparing two sets of data to be uploaded from two virtual machines into two accounts respectively. Each set was composed of multiple preselected files, whose content were identical to the corresponding file in the other set. These files were categorized into several groups for different testing purposes. A list of these files is in Table 3-2:

Table 3-3 Experiment I file list 1

Group	File list (acnt1A and acnt1B)
upload	upload1.docx upload2.xlsx upload3.jpg
view	view1.docx view2.xlsx view3.jpg
download	download1.docx download2.xlsx download3.jpg
trash	trash1.docx trash2.xlsx trash3.jpg
delete	delete1.docx delete2.xlsx delete3.jpg
open	open1.docx open2.xlsx open3.jpg
open and edit	edit1.docx edit2.xlsx edit3.jpg edit4.jpg
re-upload	re-upload1.docx re-upload2.mp3 re-upload3.pdf

To test the sharing feature of Google Drive, another two sets of files were prepared. But unlike the previous one, these two sets of files were different by filename and content.

Table 3-4 Experiment I file list 2

Group	File list	
	acnt1A	acnt1B
share	share1_1A.docx share2_1A.xlsx share3_1A.jpg	share1_1B.docx share2_1B.xlsx share3_1B.jpg
share and edit	share_edit_1A.docx	share_edit_1B.docx
share and re-edit	share_re_edit_1A.docx	share_re_edit_1B.docx

The next step was file manipulation. Metadata of all files was recorded as original metadata. Then the files were uploaded to two target Google Drive accounts by corresponding virtual machines with submission time recorded. After all files were uploaded, the following operations in Table 3-4 were conducted. Note that each account was only operated on its designated virtual machine (i.e. vm1A operated acnt1A, vm1B operated acnt1B). System time of the virtual machine was recorded for each operation. There were some time intervals between two operations.

Table 3-5 Experiment I file manipulation

Group	**Operations on files**	
	acnt1A	**acnt1B**
upload	Do nothing.	
view	View file in browser.	
download	Download files into a local folder.	
trash	Use “remove” option to move files to trash.	
delete	First remove files. Then delete it permanently from trash.	
open	Use “Open with…” function to open the file with a provided app. Then close the file.	
open and edit	edit1.docx: Open in browser with Google Docs and edit.	
	edit2.xlsx: Open in browser with Google Sheets and edit.	
	edit3.jpg: Open in browser with Pixlr Editor and edit. Then choose “Save”, not as a copy.	
	edit4.jpg: Open in browser with Pixlr Editor and edit. Then choose “Save as copy”.	
re-upload	re-upload1.docx: Use Microsoft Word to modified local file and then upload again.	
	re-upload2.mp3: Upload another edit4.mp3 with different content.	
	re-upload3.pdf: Upload the exact same file again via browser	
share	Share to acnt1B.	Share to acnt1A.
share and edit	1. Right click, select “Open with” > “Google Docs” to create Google Docs files share_edit_1A_GDocs and share_edit_1B_GDocs.	
	2. Share share_edit_1A to acnt1B.	2. Share share_edit_1B to acnt1A.
	3. Edit shared share_edit_1B.	3. Edit shared share_edit_1A.
share and re-edit	Step 1-3 similar to “share and edit” group.	
	4. Edit share_re_edit_1A again.	4. Edit share_re_edit_1A again.

Note in the table above, “view” and “open” were two different operations. In this experiment, viewing a file meant double clicking an entry of the file in the Google Drive web interface. The Google Drive web application then displayed the content of the file in view mode. Opening a file was the act of right-clicking an entry then selecting “Open with” option. Google Drive would list some apps which were able to open this type of file and read data within.

Many apps are provided by Google Drive now to allow users manipulate their files. Each app can be considered as a Software as a Service (SaaS). There is a possibility that more apps are still under development and would be offered in the future. It was unlikely to test every app which could be used in Google Drive. In this experiment, Google Docs, Google Sheets and Pixlr Editor were chosen to open .docx, .xlsx and .jpg files respectively.

After all operations were done, two approaches, namely web browser and API application, were employed to collect evidence from two accounts respectively.

On vm1A, log into Google Drive account acnt1A using web browser. For each entry in the account, select it and expand the details panel in the interface. Record all displayed information. After each entry was examined, download all files into a designated folder in vm1A. There are two methods to download files. One is batch download, where multiple files are zipped and downloaded as one zipped file. The other is normal download, where each file is downloaded individually. In this experiment, both methods were used to download all files.

On vm1B, run the API application. Connect the program to acnt1B. It will then communicate with the Google Drive API and gather all information. All information was output to a file as metadata record. After that, it downloaded all files into a folder in vm1B.

Next, FTK imager was used to generate both virtual machines’ disk images. Images were then analyzed to collect metadata of files downloaded in two virtual machines. Combined with previous obtained metadata through browser and API application, the final result of collected information was composed and recorded in a sample table as below.

Table 3-6 Sample result table

	Browser			**API application**	
	web interface	file downloaded via browser (batch)	file downloaded via browser (individual)	metadata record	file downloaded via API application
filename					
MD5 hash					
created time					
last modified time					
…					

3.4 Experiment II

Experiment II was designed to study evidence collection when client software is installed. Compared with browser, the user of client software has less control over the account. Unlike browsers, where the user manually performs all operations, client software does synchronization automatically, as long as the target account is connected. This automatic synchronization feature should be tested.

Two different virtual machines named vm2A and vm2B were created. Two free Google Drive accounts acnt2A and acnt2B were set up with no content added. The official Google Drive software was then installed and connected to the respective Google Drive accounts. The software selected a folder as local repository folder, within which all files will be synchronized with the connected account.

The following files were then copied to the local repository folders in both machines:

Table 3-7 Experiment II file list

Group	**File list**
upload	upload1.docx upload2.xlsx upload3.jpg
view	view1.docx view2.xlsx view3.jpg
delete	delete1.docx delete2.xlsx delete3.jpg
edit	edit1.docx edit2.xlsx edit3.jpg
share	share1A(B).docx share2A(B).xlsx share3A(B).jpg
synchronize	sync1.docx sync2.jpg sync3.mp3

The client software would automatically upload all files to connected accounts. Wait after a certain period to make sure all files are uploaded. Create two more virtual machines vm2Async and vm2Bsync. For each machine, install client software. Connect client software in vm2Async to account acnt2A and vm2Bsync to acnt2B. Let the two machines synchronize all files and turn them off. These two new virtual machines are used later to create synchronization scenarios.

Then go back to vm2A and vm2B, apply the following operations on both machine while connected to corresponding accounts. Note all operations are performed in local repository folder within each virtual machine. All time of operations is recorded.

Table 3-8 Experiment II file manipulation 1

Group	Operations
upload	Do nothing.
view	Open and view with default program of operating system and close.
delete	Delete files.
edit	Open edit1.docx, edit, then save.
	Rename edit2.xslx into edit2_rename.xslx
	Copy another edit.jpg with completely different content to the folder, overwriting the previous edit.jpg
share	Share files to the other account.

After the operations above, turn off vm2A and vm2B. Turn on vm2Async and vm2Bsync. Perform the following operations in their local repository folders:

Table 3-9 Experiment II file manipulation 2

synchronize	Open sync1.docx, delete some content, then save.
	Rename sync2.jpg into sync2_rename.jpg
	Copy another sync3.mp3 with completely different content to the folder, overwriting the previous sync3.mp3
	Create sync4.docx, type some content and save it. Wait after synchronization completes, then delete it from the folder.

With all the above done, the virtual machines and target accounts will be processed as below:

Without turning the vm2A on, make disk image of it with FTK imager. Save the image as vm2Aoff.001.

Load vm2A in virtual machine software and turn it on. Internet connection for the virtual machine is provided to allow synchronization. After synchronization finished, turn it off again and make disk image as vm2Aon.001.

Create a new virtual machine vm2C. In this machine, connect the written API program to acnt2B and collect information. All gathered information is exported to metadata record. After that, it downloads all files into a folder.

After intervention is observation. Load vm2Aon.001 and vm2Aoff.001 in FTK. Record all obtained information.

Chapter 4: Results and Analysis

4.1 Experiment I results

This part summarizes and analyzes results obtained by experiment I. The results are divided into several categories to illustrate different aspects of evidence collection and data integrity.

4.1.1 Basic file information

Table 4-1 Basic information

	Browser			**API application**	
	web interface	file downloaded via browser (batch)	file downloaded via browser (individual)	metadata record	file downloaded via API application
filename	as original	as original	as original	as original	changeable
file size	as original	as original	as original	as original	as original
MD5 hash	n/a	as original	as original	as original	as original
MIME type	n/a	as original	as original	as original	as original

The result shows that almost all basic file information was kept if the files were downloaded either via browser or API application. The Google Drive web interface did not provide a page to display MD5 hash value or MIME type of a file. Note that the filename of a file downloaded via API was determined by the codes. The programmer can use any legal filenames to write acquired data stream into files.

4.1.2 File user information

Table 4-2 File user information

	Browser			**API application**	
	web interface	file downloaded via browser (batch)	file downloaded via browser (individual)	metadata record	file downloaded via API application
owner	yes	n/a	n/a	yes	n/a
sharing user	yes	n/a	n/a	yes	n/a
current user permission	yes	n/a	n/a	yes	n/a
last modifying user	yes	n/a	n/a	yes	n/a

The user information metadata only existed when the file was stored in Google Drive account. Once a file was downloaded, the user information was removed. All user information could be checked by right-clicking an entry in Google Drive web interface, selecting "Share…" option and clicking the "Advanced" button. Drive API could retrieve part of user information

4.1.3 Deleted files recovery

Table 4-3 Deleted files recovery

	Browser			**API application**	
	web interface	file downloaded via browser (batch)	file downloaded via browser (individual)	metadata record	file downloaded via API application
trashed file	yes	yes	yes	yes	yes
deleted file	no				

Files removed in Google Drive web interface were moved to trash. A trashed file was possible to be restored. To use browser to restore a trash file, first find it in the "Trash" page, then select the file and use "Restore" option. A trashed file was unable to be downloaded via browser directly unless being manually recovered. On contrary, Drive API allowed an application to download trashed files directly without restoration. All trashed files download by API have the “trashed” property set as true in metadata record.

If a file was deleted from trash, it was considered to be removed permanently. There was no known method to recover it via either browser or API.

4.1.4 File opening and editing via browser

Before discussing collected information about editing via two approaches, the outcomes of these operations were listed as below.

Table 4-4 Outcome of file opening and editing via browswer

Group	File	Outcome
open	open1.docx	A new entry was created by Google Docs.
	open2.xlsx	A new entry was created by Google Sheets.
	open3.jpg	No obvious change
open and edit	edit1.docx	The new entry was edited by Google Docs.
	edit2.xlsx	The new entry was edited by Google Sheets.
	edit3.jpg	The original jpg file content was modified.
	edit4.jpg	A new jpg file was created. Modification only appeared in the new file.

Results in table 4-4 suggested that the outcome of opening and editing of files stored in Google Drive depended on the app used. In experiment I, the Google Drive web application allowed Google Docs or other authorized apps to open and edit files. Once a file was opened (not viewed), a new entry containing data from original file was created by the app. All subsequent editing will be performed over this entry. On contrary, when using Pixlr Express, a user could choose whether to save the modification to the original file or as a new copy.

Thus, the outcome of opening and editing is unpredictable. Different apps may generate different amount of evidence during the opening and editing process.

4.1.5 Upload mechanism of Google Drive web application

Table 4-5 Outcome of re-uploading files

Group	File	Outcome
re-upload	re-upload1.jpg	A new re-upload1.jpg was created.
	re-upload2.mp3	A new re-upload2.mp3 was created.
	re-upload3.pdf	A new re-upload3.pdf was created.

The upload mechanism of Google Drive web application was more straightforward than normal file manager programs in regular operating systems. When a file was uploaded via browser, Google Drive simply created a new entry to store it. It did not check whether a file with same filename or same content had existed already, let alone replaced the old file with a new one.

If the user downloads multiple files in the same name simultaneously, they will be zipped into one file. In the zipped file, different postfixes will be added to original filenames to distinguish them. For example, FILENAME.txt and FILENAME(1).txt. Note the later one is not necessarily uploaded or created after the first one.

4.1.6 Timestamps

All data related to date and time from Google Drive API is stored in RFC 3339 timestamp format. Every timestamp record ends with a “Z”, which, according to RFC 3339, indicates the time is Coordinated Universal Time (UTC) with no offset (Network Working Group, 2002). So when dealing with date and time from Google Drive API, there is no information about time zones where the file was uploaded, viewed or edited.

Timestamps of files stored in Google Drive can be changed by many file operations. This section discusses timestamps in different cases.

Table 4-6 Timestamps of files downloaded to local system

	Browser		**API application**
	file downloaded via browser (batch)	file downloaded via browser (individual)	file downloaded via API application
created time	downloaded time	downloaded time	downloaded time
modified time	two hours prior to uploaded time	downloaded time	downloaded time
last access time	downloaded time	downloaded time	downloaded time

When a file was downloaded from Google Drive, all timestamps were changed significantly. Table 4-4 showed the result. Almost all timestamps were set as the time when the file is downloaded, no matter how the file was manipulated in Google Drive. The only exception was batch downloaded files had modified time marked as two hours prior to their uploaded time. The cause of this phenomenon is still unclear. Result of experiment I indicated that downloaded files had altered timestamps.

The rest of this part discusses timestamps acquired via the web interface of browser and API application after different file manipulations.

Table 4-7 Timestamps of uploaded only files

	browser web interface	**API application metadata record**
created time	uploaded time	uploaded time
modified time	uploaded time	uploaded time
modified by me time	uploaded time	uploaded time
last viewed by me time	n/a	n/a

If a file was uploaded by browser only, the created time and modified time are set as time of upload. Both can be obtained either by web interface or Drive API. There was no value set for access time.

Table 4-8 Timestamps of viewed files

	browser web interface	**API application metadata record**
created time	uploaded time	uploaded time
modified time	uploaded time	uploaded time
modified by me time	uploaded time	uploaded time
last viewed by me time	viewed time	viewed time

When a file was viewed in web interface, the viewed time was recorded. However, Google Drive only recorded the time of file being viewed by current user. There was no way to tell whether some other users viewed a file if the file was shared among multiple users.

Table 4-9 Timestamps of downloaded files

	browser web interface	**API application metadata record**
created time	uploaded time	uploaded time
modified time	uploaded time	uploaded time
modified by me time	uploaded time	uploaded time
last viewed by me time	n/a	n/a

All timestamps were identical to those of an uploaded only file. This result indicted that the timestamps of files stored in Google Drive would not change no matter how the file was downloaded. In other words, the download process of a file was not considered as a view (or open) action.

Table 4-10 Timestamps of opened and edited file (direct modification)

	browser web interface	**API application metadata record**
created time	uploaded time	uploaded time
modified time	edited time	edited time
modified by me time	edited time	edited time
last viewed by me time	edited time	edited time

Table 4-11 Timestamps of opened and edited file (create new entry)

	browser web interface		**API application metadata record**	
	original file	**new entry**	**original file**	**new entry**
created time	uploaded time	opened time / edited time	uploaded time	opened time / edited time
modified time	uploaded time	edited time	uploaded time	edited time
modified by me time	uploaded time	edited time	uploaded time	edited time
last viewed by me time	viewed time	edited time	viewed time	edited time

As section 4.1.4 explained, open and edit operations may modify original file directly or create a new entry of the original file. In the former case, timestamps of original file were changed. In the latter case, original file was unchanged. Timestamps of new entry created by other apps depended on the design of the app, more specifically, the time when the app writes data. Some apps, such as Google Docs, write data to new file as soon as it opens a file and write automatically and instantly if any modification is detected while some other apps do not unless it is told to do so explicitly by user.

Table 4-12 Timestamp of shared files

	browser web interface	**API application metadata record**
created time	uploaded time	uploaded time
modified time	shared time	shared time
modified by me time	shared time	shared time
last viewed by me time	n/a	n/a

The act of merely sharing a file was considered as a modification to the file, causing a change in the modified time. But it was not considered as a view operation. Thus, the owner of the shared file can see both modified time and “modified by me” time. But other user who sees this shared file can only see modified time if the user does not perform any modification.

4.1.7 Revisions

Revisions of a file were viewed via browser. Google Drive web interface offered a “Manage versions…” option to view revision history for most types of files. It listed all previous versions, along with corresponding modified time and modifying user. Download link to each version was also provided.

More detailed revision records could be obtained by Google Drive API. By requesting the API, an application could acquire these records in JSON format. Extra information offered in detailed record included revision ID, MD5 checksum, file size and others.

According to collected revision records in this experiment, each time a file was uploaded, created, or modified, a revision record was created. Thus, besides entries that pointed to folders, each entry had at least one revision record. As mentioned earlier, Google announced the maximum period of revisions being kept as 30 days. Nevertheless, during this experiment, some revisions older than six months were still found and recovered successfully.

Information encapsulated in revision records made them extremely useful when trying to build timelines for file modifications. It is likely for examiners to know who did what and when by looking at the revision records.

4.1.8 Summary

Experiment I revealed how data was collected via web browser and API application. Both approaches provide methods to download files. The data integrity of downloaded file’s content was verified by hash values. However, the integrity of metadata was not achieved.

Google Drive employs a system where rich metadata is embedded with files stored inside. This metadata can be easily changed or ripped off during download process. But by using web interface and Drive API, these metadata can be extracted to assist digital forensic investigation.

4.2 Experiment II results

This part summarized and analyzed results obtained by experiment II.

4.2.1 vm2Aoff.001

Table 4-13 vm2Aoff.001 analysis result

group	file list	found?	created time	modified time
upload	upload1.docx upload2.xlsx upload3.jpg	yes	as original	as original
view	view1.docx view2.xlsx view3.jpg	yes	as original	as original
delete	delete1.docx delete2.xlsx delete3.jpg	recoverable	as original	as original
edit	edit1.docx (old version)	no	n/a	n/a
	edit1.docx (new version)	yes	as original	edited time
	edit2.xlsx	no	n/a	n/a
	edit2_rename.xlsx	yes	as original	as original
	edit3.jpg (old version)	no	n/a	n/a
	edit3.jpg (new version)	yes	as original	modified time of new version
share	share1B.docx share2B.xlsx share3B.jpg	no	n/a	n/a
synchronize	sync1.docx (old version)	yes	as original	as original
	sync1.docx (new version)	no	n/a	n/a
	sync2.jpg	yes	as original	as original
	sync2_rename.jpg	no	n/a	n/a
	sync3.mp3 (old version)	yes	as original	as original
	sync3.mp3 (new version)	no	n/a	n/a
	sync4.docx	no	n/a	n/a

Every file’s accessed time was same as its created time, which was the time it was copied to the client software repository folder. The only exception was edit1.docx (new version), whose accessed time was same as its edited time.

Result of Table 4-10 indicated the possibility of recovering most data from client software. However, this method was unable to track version changes, including direct edits or overwriting file with another file with same filename.

It was also reasonable that client software could not synchronize changes in Google Drive account when the machine was off or disconnected from the internet.

4.2.2 vm2Aon.001

Table 4-14 vm2Aon.001 analysis result

group	file list	found?	created time	modified time
upload	upload1.docx upload2.xlsx upload3.jpg	yes	as original	as original
view	view1.docx view2.xlsx view3.jpg	yes	as original	as original
delete	delete1.docx delete2.xlsx delete3.jpg	recoverable	as original	as original
edit	edit1.docx (old version)	no	n/a	n/a
	edit1.docx (new version)	yes	as original	edited time
	edit2.xlsx	no	n/a	n/a
	edit2_rename.xlsx	yes	as original	as original
	edit3.jpg (old version)	no	n/a	n/a
	edit3.jpg (new version)	yes	as original	modified time of new version
share	share1B.docx share2B.xlsx share3B.jpg	no	n/a	n/a
synchronize	sync1.docx (old version)	no	n/a	n/a
	sync1.docx (new version)	yes	as original	edited time of another VM
	sync2.jpg	no	n/a	n/a
	sync2_rename.jpg	yes	as original	edited time of another VM
	sync3.mp3 (old version)	no	n/a	n/a
	sync3.mp3 (new version)	yes	as original	edited time of another VM
	sync4.docx	no	n/a	n/a

Image vm2Aon.001 was analyzed to compare with vm2Aoff.001. The differences were in the synchronize group. After the virtual machine was turned on and connected to the internet, all files were synchronized to latest versions. The accessed times of synchronized files were identical as their edited times.

The important part to be notice was the absence of sync4.docx in both vm2Aon.001 and vm2Aoff.001. Because this file was uploaded and deleted by another client during the time when virtual machine was disconnected, the client software had no clue about its existence, thus left no trace about it.

4.2.3 API application metadata record

Table 4-15 API application metadata record

group	file list	found?	created time	modified time
upload	upload1.docx upload2.xlsx upload3.jpg	yes	as original	as original
view	view1.docx view2.xlsx view3.jpg	yes	as original	as original
delete	delete1.docx delete2.xlsx delete3.jpg	yes	as original	as original
edit	edit1.docx (old version)	yes	as original	as original
	edit1.docx (new version)	yes	as original	edited time
	edit2.xlsx	yes	stored in edit2_rename.xlsx	
	edit2_rename.xlsx	yes	as original	as original
	edit3.jpg (old version)	yes	as original	as original
	edit3.jpg (new version)	yes	as original	new version time
synchronize	sync1.docx (old version)	yes	as original	as original
	sync1.docx (new version)	yes	as original	edited time of another VM
	sync2.jpg	yes	stored in sync2_rename.jpg	
	sync2_rename.jpg	yes	as original	edited time of another VM
	sync3.mp3 (old version)	yes	as original	as original
	sync3.mp3 (new version)	yes	as original	edited time of another VM
	sync4.docx	yes	as original	as original

Table 4-12 listed the API application metadata record. The API application acquired all versions of files, regardless how different machines synchronized. It is worth noting that when a file was renamed, its new filename was set as title. But the old filename was still stored in originalFilename property of the file.

4.2.4 Revisions

Unlike web interface, the Google Drive client software did not support viewing or recovering revisions. Nevertheless, Google Drive API was still applicable to be used to retrieve revision records stored in Google Drive as described in 4.1.7 in experiment I.

4.2.5 Summary

Client software keeps synchronizing the latest version of file as long as internet connection is available. However, it does not maintain old versions that were overwritten by synchronization, meaning it can not acquire evidence before and after synchronization simultaneously. If a file is uploaded then deleted by others when current machine is offline, the change will not be caught by client software.

The experiment also indicates that client software does not track files which are shared to the current user but not owned by current user. Only files owned by current user are downloaded by client software.

On contrary, Drive API keeps tracks of all versions and information of files. All files, including owned and shared ones, and metadata record could be retrieved.

Chapter 5: Discussion

5.1 Research Problem

As the number of cloud storage users keeps growing, digital forensic examiners are preparing for the incoming challenges brought by cloud storage services. As one of the most popular cloud storage services today (Griffith, 2014), Google Drive becomes a valuable object to study. Quick and Choo (2014) and Federici (2014) experimented different approaches to collect evidence from Google Drive, but still leaving gaps.

To solve the problem, this research utilized Google Drive API as the tool to collect evidence from Google Drive. The results were compared with web browser and client software approaches to find out the recommended one.

5.2 Answers to Research Questions

This section compares three approaches: web browser, client software and API. The goal of the comparison is to answer the two research questions raised in section 1.4.

5.2.1 Answer toQ1

Research question Q1 was "What types of evidence in Google Drive can be found via the API approach?"

There is little difference among these three approaches when they are applied to download current files stored in a Google Drive account. All three approaches can download current files effectively, as long as internet connection is available.

When dealing with deleted files and file revisions, web browser and API approaches have significant advantages over client software. The former two approaches could find all these files recorded by Google Drive file system easily. On contrary, client software can only recover deleted files with the help of forensic software while had nothing can do to restore older versions.

The Google Drive provides rich metadata properties for files stored within. This study reveals that most of these properties can only exist within Google Drive. Once the files are downloaded, they are ripped off from the files. By using client software, the very basic metadata with forensic values such as timestamps can be obtained. However, through web browser, the detail panel in Google Drive web interface provides much richer metadata, including owners, sharing status, activities and so on. Nevertheless, the metadata given by web browser is still too few compared to API approach. Google Drive API can request JSON record of the file, which contains all properties of the file.

In addition to the properties mentioned above, some other forensically valuable properties currently provided only by Google Drive API include:

originalFilename: This is the original filename while a file is uploaded to Google Drive. User of Google Drive can rename files at any time. But renaming operation only changes the **title** property of the file. The original filename is stored in this property and is unchangeable.

md5Checksum: It stores the MD5 hash value of file content. It is very useful when comparing the integrity of files downloaded and files stored in cloud. It also allows examiners to locate possible “flagged” files with known MD5 hash value quickly.

imageMediaMetadata: This group of properties contains information of photos taken by cameras, usually known as Exif(Exchangeable image file format). It provides critical information of photos and cameras, including date and time, manufacturer, model, or even detailed latitude and longitude of the location where the photo was taken for some advanced cameras.

extended properties: Although Google Drive already provides rich properties for entries, Google allows developers of Google Drive related applications to add extended properties to entries in the form of “key-value” pairs. For example, a word editor may add a “wordCount” property to each text file it opens to record the number of words in this file. This kind of property is not visible directly through browser but can be requested via API.

In summary, the API approach acquires the most evidence compared with other two methods.

5.2.2 Answer to Q2

Research question Q2 was “How was the integrity of evidence maintained during the acquisition via the API approach?”

Maintaining good evidence integrity is the goal of all forensic investigations. The data integrity can be categorized into two: file content integrity and metadata integrity.

The result of experiment I and II indicated file content integrity is guaranteed by Google Drive services. All downloaded files, regardless how they are downloaded, shared same MD5 hash values with their original copies. This result is reasonable. After all, no regular customers would like to use a cloud storage service that could not even guarantee the integrity of their saved data content.

Metadata integrity is much more complicated. There are many operations on files that could alter metadata. This research indicates that the timestamps of files downloaded by either web browser or API application are altered heavily, not to mention the other metadata lost during the process. On the other hand, client software does a good job on keeping important timestamps by auto synchronization in relative to downloading manually.

The most effective ways to collect metadata are still web interface through web browser or API. The most useful metadata is stored within the Google Drive account and usually does not come along with the files when they are downloaded. Results of experiment I and II shows that such metadata collected via these two approaches can reflect what happened to the files in most cases. For example, the “modifiedByMeDate” can tell the time when the current user modified this file recently. Something worth mention is that downloading files does not alter metadata of files in Google Drive.

In summary, the API approach maintained well evidence integrity, as long as using metadata stored in Google Drive as reference.

5.2.3 Other factors

Other than the two important points above, some other factors should also be used to evaluate the API approach.

One factor is efficiency. According to the previous discussion, it looks like the web browser approach works as well as API approach. However, to acquire most information with browser approach, a forensic examiner has to open the Google Drive account and select each file one by one manually to check its metadata. All trashed files and old versions must be manually restored before being downloaded. Such a manual process is both inefficient and error prone. For example, an examiner may accidentally double-click a file instead of single-click it, which leads to an unintended “open” action, compromising evidence integrity. This could be a hard problem to examiners especially when the amount of stored files is huge. On contrary, the API application allows all processing be done by automation. A well-designed program can handle all data in a fast, accurate and comprehensive manner. As the amount of data to be analyzed by digital forensic examiners is growing larger and larger, an automatic process is always welcomed by examiners.

A second factor is authentication. There exists possibility that a court order is issued but no username and password is obtained to access Google Drive. The API approach can solve this problem in some cases. As section 3.2.1 explained, it was possible to bypass the authentication process if cookies are stored in the browser in order to avoid the trouble of typing password each time. Generally speaking, if web browser approach could be used to access Google Drive, so could the API.

The last factor is the required knowledge and skills for each approach. This is where the API not good at. API can not be used directly to acquire evidence. Codes must be written to utilize API. Luckily, this problem can be solved by letting professional forensic software companies develop software based on Google Drive API and release to the market. Another solution is preparing a simple piece of codes written in a script programming language, such as Python or Ruby, so that the codes can easily run on target machine without being compiled first.

5.3 Principles of applying API approach

As introduced previously, the API approach is powerful when gathering evidence from Google Drive. Nevertheless, if not used correctly, it may still create false results. Here are some suggested principles while applying API approach in real world.

5.3.1 Identify files by file ID

In traditional digital forensic examination, a file is identified by the combination of "folder path + filename" because such a combination is unique for each file. However, such pattern does not work that well in the case of Google Drive. In Google Drive, all entries in an account are stored in a single container. Although users are allowed to create folders and put entries into different folders, these folders only create a virtual directory structure. The entries are virtually stored in the same level. That is why the API can request all entries by merely visiting the "root folder" without knowing the directory structure. On the other hand, each entry in Google Drive has a property named "title", which is usually generated from the filename of a file when it is uploaded. Unlike common operating systems, Google Drive allows different entries use a same title, making the title not necessarily unique for each file. The actual unique value Google Drive uses to identify an entry is the entry's file ID. A file ID is a 28-character long string unique for each entry. When the API is requesting for a certain file, Google Drive always asks for the file ID, not the filename.

Such feature must be considered when acquiring evidence via API approach. Assume the following scenario: An application is downloading all files stored in a Google Drive account into one single folder in local hard disk. Each time a file is written, the title property of the file is used directly as filename. Unfortunately, there are two images with the same title "evidence.jpg" in the Google Drive. The first image is written to disk correctly. But when the application is trying to write the second image, things go wrong because there is already an "evidence.jpg" in the folder. If not designed well, the application may either fail to write the second image or overwrite the first image with the content of the second one.

To solve the problem, simply renaming the second image with postfix is feasible. However, a better solution is employing the file ID, which is always unique for each file. Suppose the two images are with file ID "AAA" and "BBB" respectively, filenames written to disk can be "evidence_AAA.jpg" and "evidence_BBB.jpg" so that they can both be stored and distinguished correctly.

5.3.2 Separation of file content and metadata

As explained previously, metadata stored within the Google Drive account does not always come along with the files when they are downloaded. The metadata of files downloaded via API is neither comprehensive nor valid. Thus, when analyzing files in Google Drive, metadata of downloaded files should never be relied on.

The metadata created by Google Drive preserves excellent information for forensic examination. Requesting metadata of a file is an action independent of requesting its content. Both actions should be performed by the application in order to provide maximum information to examiners. Because file contents and metadata are requested via file ID, the file ID value can be used to link a file's content and metadata together to create a data structure describing the file as evidence. In addition to metadata, some other related resources, for example, revisions should be processed in a same manner.

Figure 5-1 illustrates how the evidence is acquired and recorded.

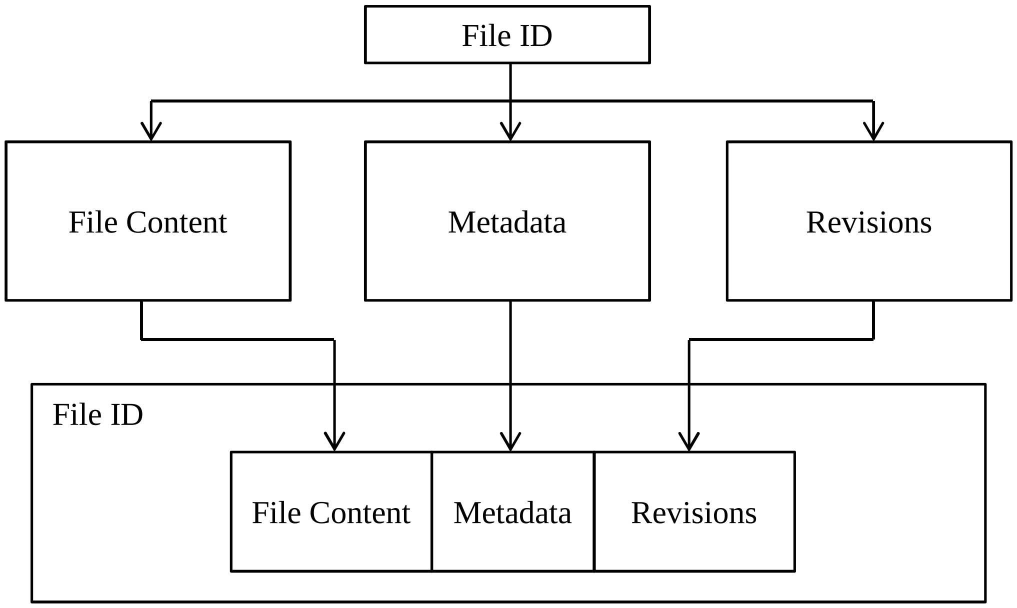


Figure 5-1 Evidence Acquisition and Recording

5.3.3 Mind the volatility of evidence

One of the most significant differences between evidence in traditional storage media and in cloud is the volatility of the latter. Although the API approach has been proved effective in preserving evidence integrity, since digital forensic examiners do not have the control of hardware, it is still possible that evidence stored in Google Drive be modified by someone else during investigation. For example, someone could use the username and password to login to the suspect’s account and delete evidence permanently. The suspect’s permission to view a shared file in another account may be cancelled by the owner. These are events that examiners unlikely to prevent.

For this reason, the examination over evidence in Google Drive should be conducted as soon as possible. The later the examination is, the more likely the evidence will be tampered, even if the suspect machine is already in custody. When conducting the examination, all evidence stored in the Google Drive account, including file content, metadata and revisions should be downloaded to allow static and local analysis instead of live and remote analysis. If the total size of files is large, it is even possible that the files have been changed by someone other than the examiner during the download process. Each time an examination is conducted, the date and time of the examination must be recorded in a log. If the Google Drive account is suspected to be still being used by someone, repeated examination can be performed to acquire new evidence added.

In summary, keep in mind that the acquired evidence does not represent a permanent state of the Google Drive account, but the temporary state at the time of examination.

5.4 Concerns about security

According to the results of the experiments, application based on Google Drive API is a powerful tool when applied in digital forensic examination. However, this tool is possible to be abused by malicious attackers whose intention is gathering user data for illegal purposes. By using such a tool, an attacker may download hundreds of files and related metadata in just a few minutes.

To prevent such things from happening, Google requires all applications connected to the

Google Drive to be authorized by the owner of the account. Nevertheless, by combining with some social engineering techniques, it is possible for attackers to trick the owner into giving them the authorization without knowing their real purposes.

Once an application is connected to a Google Drive account, it will be difficult to detect its malicious activities. As the experiment results showed, an application using Google Drive API can easily acquire all data without changing any of them if it wants to. This means no trace is left behind. A user may not know how many files the application has visited or downloaded, making the user unaware of his or her stored data is being stolen. One possible countermeasure is the Apps resource offered by Google Drive API. It lists names of all applications installed or connected to this account along with some other information (Google, 2015b). Users can review the list to determine if there are applications they are not expecting. Although such information may be carefully crafted by attackers to make the application looked like a decent tool, for example, a file manager.

5.5 Limitations

Due to the rapid development of cloud computing, the number of cloud storage services grows in great speed. New service providers keep emerging and joining the battle to scramble market shares. There is no written standard of API for all cloud storage providers, although some non-mandatory universal principles do exist and are followed by most providers. In such an environment, it is extremely difficult to develop a universal forensic tool to cover all cloud storage services while using all capacity of each one.

Even when considering a single cloud storage service provider, problems still exist. To face out fierce competition, service providers keep updating their products and expanding existing services. API of a cloud storage service is regularly expanded to provide more features, sometimes even be updated into a whole new version. Luckily, these kinds of updates are usually downward compatible. That is, programs based on old version API can still run in new API. Nevertheless, forensic tools developed based upon API are very likely to fall behind normal software supported by service providers themselves, making them less effective.

Finally, this research focuses on personal used cloud storage services. Besides that, there are cloud storage services offered to business users from small companies to large enterprises. Business cloud storage service has much higher level of scale, complexity and customized features. The measures applied on personal cloud storage service may still be able to be used on business, but not so effective.

5.6 Future works

This research mainly focused on collecting basic file information from Google Drive. But the Google Drive API also offers other hidden features. For example, Google Drive API provides Comments and Replies resources, which enable leaving comments to a certain file (Google, 2015a). At the time of writing, this function is still unavailable in Google Drive web interface. But since it is in Google Drive API reference, it is usable by other applications. Forensic examiners and forensic software developers should keep an eye on such features to fully discover the potential of Google Drive API.

To deal with the security risks brought by Google Drive API, developers and security experts should work on countermeasures against possible attacks. Although the possibility of being employed as a weapon of attack exists, Google Drive API can also be used as a shield to defend data privacy and integrity. Applications that use Google Drive API should utilize its functions and add extra features such as logging to help digital forensic examiners when incidents occur.

5.7 Conclusion

The digital forensic community has been experiencing rapid changes recent years. Concepts such as cloud computing, mobile business and big data are sweeping the information technology industry in stunning speed. It is critical for digital forensic examiners to equip themselves with advanced tools and ideas to overcome the challenges brought by these developments.

This research discussed using API approach to acquire evidence stored in Google Drive, a popular cloud storage service on the market. Current development of Google Drive and some other cloud storage forensics were discussed. After analyzing the features of Google Drive API, an application was written in order to meet the requirements consisted of authentication, data integrity, data acquisition and output.

Experiments were conducted in the following part aimed to discover the forensic value of API. The analysis and discussion about obtain result suggested the API approach was suitable to be employed to conduct digital forensic investigation.

Findings of this research can help digital forensic examiners choose the most convenient tool to perform their tasks. The author also hopes these findings could become part of the guidelines for software engineering when developing forensic software targeting cloud storage forensics.

References

Chung, H., Park, J., Lee, S., & Kang, C. (2012, 11). Digital forensic investigation of cloud storage services. *Digital Investigation, 9*, pp. 81-95.

Digital Corpora. (2014). *Digital Corpora » Govdocs1*. Retrieved from Digital Corpora: http://digitalcorpora.org/corpora/govdocs

Duke, A. (2014, 10 12). *5 Things to know about the celebrity nude photo hacking scandal*. Retrieved from CNN.com: http://www.cnn.com/2014/09/02/showbiz/hacked-nude-photos-five-things/

Dykstra, J., & Sherman, A. T. (2012). Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques. *Digital Investigation, 9*, pp. 90-98.

Federici, C. (2014). Cloud Data Imager: A unified answer to remote acquisition of cloud storage areas. *Digital Investigation, 11*, pp. 30-42.

Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* Doctoral dissertation, University of California, Irvin.

Google. (2014, 6 25). *Meet the new Google Drive*. Retrieved 8 20, 2014, from Google Drive Blog: http://googledrive.blogspot.com/2014/06/meet-new-google-drive_25.html

Google. (2015, 9 1). *API Reference | Drive REST API | Google Developers.* Retrieved 12 5, 2015, from Google Developer: https://developers.google.com/drive/v2/reference/

Google. (2015, 3 19). *Apps: list | Drive REST API | Google Developers.* Retrieved 12 5, 2015, from Google Developers: https://developers.google.com/drive/v2/reference/apps/list

Google. (2015, 10 5). *Files | Drive REST API | Google Developers*. Retrieved 12 1, 2015, from Google Developers: https://developers.google.com/drive/v2/reference/files#resource

Google. (2015, 4 7). *Using OAuth 2.0 to Access Google APIs*. Retrieved from Google Developers: https://developers.google.com/accounts/docs/OAuth2

Google. (n.d.). *Using Google Drive - New Features, Benefits & Advantages of Google Cloud Storage*. Retrieved 4 5, 2015, from Google Drive: https://www.google.com/drive/using-drive/

Griffith, E. (2014, 11 6). *Who's winning the consumer cloud storage wars?* Retrieved from Fortune: http://fortune.com/2014/11/06/dropbox-google-drive-microsoft-onedrive/

Internet Engineering Task Force. (2012, 10). *RFC 6749 - The OAuth 2.0 Authorization Framework.* Retrieved from IETF: http://tools.ietf.org/html/rfc6749

ITPP in association with HP. (2014, 7 18). *The cloud data storage market*. Retrieved 8 20, 2014, from ITProPortal.com: http://www.itproportal.com/2014/07/18/the-cloud-data-storage-market/

Johnston, S. (2014, 6 25). *Unlimit your business with Google Drive for Work*. Retrieved from Official Google for Work Blog: http://googleforwork.blogspot.com/2014/06/unlimit-your-business-with-google-drive.html

Lardinois, F. (2012, 10 15). *Report: Cloud Storage Services Now Have Over 375M Users, Could Reach 500M By Year-End*. Retrieved 8 9, 2014, from TechCrunch.com/: http://techcrunch.com/2012/10/15/report-cloud-storage-services-now-have-over-375m-users-could-reach-500m-by-year-end/

Marturana, F., Me, G., & Tacconi, S. (2012). A Case Study on Digital Forensics in the Cloud. *2012 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC).*

Merriman, C. (2014, 9 5). *IDC: Hard drive market is shrinking due to cloud shift*. Retrieved 9 7, 2014, from The Inquirer: http://www.theinquirer.net/inquirer/news/2363694/idc-hard-drive-market-is-shrinking-due-to-cloud-shift

National Institute of Standards and Technology. (2014). *Computer Forensics Tool Catalog - Tool Search*. Retrieved 8 10, 2014, from Computer Forensics Tool Catalog: http://www.cftt.nist.gov/tool_catalog/populated_taxonomy/index.php?all_tools=all&ff_id=20&1%5B%5D=any&2%5B%5D=any

Network Working Group. (2002, 7). *Date and Time on the Internet: Timestamps.* Retrieved 10 10, 2015, from The Internet Engineering Task Force: https://www.ietf.org/rfc/rfc3339.txt

Quick, D., & Choo, K.-K. R. (2013). Forensic collection of cloud storage data: Does the act of collection result in changes to the data or its metadata? *Digital Investigation, 10*, pp. 266-277.

Quick, D., & Choo, K.-K. R. (2014). Google Drive: Forensic analysis of data remnants. *Journal of Network and Computer Applications, 40*, pp. 179-193.

Sommerville, I. (2011). *Software Engineering* (9th ed.). Boston, MA: Addison-Wesley.

Taylor, M., Haggerty, J., Gresty, D., & Hegarty, R. (2010). Digital evidence in cloud computing systems. *Computer Law and Security Review, 26*, pp. 304-308.

Taylor, M., Haggerty, J., Gresty, D., & Lamb, D. (2011, 3). Forensic investigation of cloud computing systems. *Network Security*, pp. 4-10.

Appendix

This appendix shows a sample application used to acquire information from Google Drive account via Google Drive API. The application is written in Python 3.5 script language, meaning that it can run on multiple platforms without being compiled first. The functions of this sample application include:

- List all files in the Google Drive account
- Create a summary text file
- Retrieve and stored all metadata
- Download all files with content
- Provide revision information about each file

To run the application, the following steps must be done first:

1. Register a project in Google Developers Console
2. Obtain OAuth 2.0 credential
3. Download client secret file and save as client_secret.json in the same folder as the application
4. Install Python Google Client Library

Detailed steps can be found on Google Developers website link given below. Follow step 1and 2 in the quick start manual.

https://developers.google.com/drive/web/quickstart/python

Main.py

```
#Main.py
#Author: Shujian Yang
#University of Central Oklahoma
#Email: yang_shujian@hotmail.com

import httplib2
import os

import Files
import Revisions

from apiclient import discovery
import oauth2client
from oauth2client import client
from oauth2client import tools

try:
   import argparse
   flags = argparse.ArgumentParser(parents=[tools.argparser]).parse_args()
except ImportError:
   flags = None

#This is the minimun scope required to get file metadata and content
SCOPES = 'https://www.googleapis.com/auth/drive.readonly'

#File where the client secret of the app is stored
CLIENT_SECRET_FILE = 'client_secret.json'

APPLICATION_NAME = 'Google Drive Forensics'


def get_credentials(clear_token):
   if not os.path.exists('tokens'):
      os.makedirs('tokens')
   credential_path = os.path.join('tokens', 'acess_token.json')

   store = oauth2client.file.Storage(credential_path)
   credentials = store.get()
   if clear_token and credentials:
      store.delete()
      credentials = None
   if not credentials or credentials.invalid:
      flow = client.flow_from_clientsecrets(CLIENT_SECRET_FILE, SCOPES)
      flow.user_agent = APPLICATION_NAME
      if flags:
         credentials = tools.run_flow(flow, store, flags)
      else:
         credentials = tools.run(flow, store)
      print('Storing credentials to ' + credential_path)
   return credentials

def main():
   answer = input('Do you want to clear the previous acess token? (y/n) ')
   if answer and answer.upper()[0] == 'Y':
      credentials = get_credentials(True)
   else:
      credentials = get_credentials(False)

   http = credentials.authorize(httplib2.Http())
   service = discovery.build('drive', 'v2', http=http)
```

```
while True:
    #Print user information
    about = service.about().get().execute()
    print()
    print('===User information===')
    print('Current user name: {0}'.format(about.get('name')))
    print('Current user email: {0}'.format(
                about.get('user').get('emailAddress')))
    print('Root folder ID: {0}'.format(about.get('rootFolderId')))
    print('Total quota (bytes): {0}'.format(about.get('quotaBytesTotal')))
    print('Used quota (bytes): {0}'.format(about.get('quotaBytesUsed')))

    print()
    print('======================')
    print('Available actions:')
    print('1. List all files.')
    print('2. Create summary.')
    print('3. Download all metadata.')
    print('4. Download all file contents.')
    print('5. Check file revisions.')
    print('q. Quit program.')
    print('======================')

    answer = input('Enter your action: ')
    if answer == '1': #List all stored entries
        results = service.files().list().execute()
        items = results.get('items', [])
        if not items:
            print('No files found.')
        else:
            index = 1
            for item in items:
                print('{0}. {1}'.format(index, item['title']))
                index += 1
    elif answer == '2': #Create summary
        Files.write_summary(service)
    elif answer == '3': #Download all metadata
        Files.download_all_metadata(service)
    elif answer == '4': #Download all Files
        Files.download_all_files(service)
    elif answer == '5': #Check file revisions
        revision_summary = Revisions.list_revisions(service)
        if not revision_summary:
            print('No files found.')

        print()
        print('Listing revisions...')
        index = 1
        for entry in revision_summary:
            print('{0}. {1} => {2}'.format(index,
                        entry['entry_title'], entry['revision_count']))
            index += 1

        while True:
            print()
            print('Please enter the index of a file to downlod all its revisions.')
            print('"--or--')
            print("Enter 'q' to return to upper level menu.")
            choice = input()
            if choice == 'q':
                break
            try:
                    num = int(choice)
                except Exception:
                    print('Please enter a correct number.')
                    continue

                if num < 1 or num > len(revision_summary):
                    print('===Please enter a correct number.===')
                    continue

                #If entered number is valid
                revision = revision_summary[num - 1]
                Revisions.download_all_revisions_metadata(service,
                            revision['entry_id'], revision['entry_title'])

        elif answer == 'q':
            break

if __name__ == '__main__':
    main()
```

Files.py

```
#Files.py
#Author: Shujian Yang
#University of Central Oklahoma
#Email: yang_shujian@hotmail.com

from apiclient import errors
import os

RESULT_PATH = 'Result'

BASE_URI = 'https://www.googleapis.com/drive/v2/files/'

def write_summary(drive_service):
    """
    This method scans all entries in the Google Drive account.
    Then write important metadata of each entry to a txt file.
    """

    if not os.path.exists(RESULT_PATH):
        os.makedirs(RESULT_PATH)

    try:
        summary_path = os.path.join(RESULT_PATH, 'Summary.txt')
        summary = open(summary_path, 'wt')
        print('Writing summary...')

        #Print user information
        about = drive_service.about().get().execute()
        print('Current user name: {0}'.format(
                                about.get('name')), file=summary)
        print('Current user email: {0}'.format(
                    about.get('user').get('emailAddress')), file=summary)
        print('Root folder ID: {0}'.format(
                              about.get('rootFolderId')), file=summary)
        print('Total quota (bytes): {0}'.format(
                        about.get('quotaBytesTotal')), file=summary)
        print('Used quota (bytes): {0}'.format(
                        about.get('quotaBytesUsed')), file=summary)
        print('====================================', file=summary)
        print(file=summary)

        #List entries information
        results = drive_service.files().list().execute()
        items = results.get('items', [])
        for item in items:
            print('File ID: {0}'.format(item.get('id')), file=summary)
            print('Title: {0}'.format(item.get('title')), file=summary)
            print('Original Filename: {0}'.format(
                        item.get('originalFilename')), file=summary)
            print('Md5Checksum: {0}'.format(
                        item.get('md5Checksum')), file=summary)
            print('File Size: {0}'.format(
                        item.get('fileSize')), file=summary)
            print('MIME type: {0}'.format(
                        item.get('mimeType')), file=summary)
            print('Created Date: {0}'.format(
                        item.get('createdDate')), file=summary)
            print('Modified Date: {0}'.format(
                        item.get('modifiedDate')), file=summary)
            print('Last Modifying User: {0}'.format(
                    item.get('lastModifyingUserName')), file=summary)
            print('Last Viewed By Me Date: {0}'.format(
                    item.get('lastViewedByMeDate')), file=summary)

            print(file=summary)
            print('Shared: {0}'.format(
                        item.get('shared')), file=summary)
            if 'sharingUser' in item:
                print('Sharing User: {0}'.format(
                    item.get('sharingUser').get('displayName')), file=summary)
            print('Last modified by me: {0}'.format(
                        item.get('modifiedByMeDate')), file=summary)
            print('Download URL: {0}'.format(
                        item.get('downloadUrl')), file=summary)
            print('Explicitly Trashed: {0}'.format(
                        item.get('explicitlyTrashed')), file=summary)

            print(file=summary)
            print('User Permission:', file=summary)
            print('Name: {0}'.format(
                    item.get('userPermission').get('name')), file=summary)
            print('Role: {0}'.format(
                    item.get('userPermission').get('role')), file=summary)
            print('Type: {0}'.format(
                    item.get('userPermission').get('type')), file=summary)

            print(file=summary)
            print('====================================', file=summary)
            print(file=summary)

            print('{0} recorded in summary'.format(item.get('title')))
    except errors.HttpError:
        print('An error occurred while downloading metadata.')
    except IOError:
        print('An error occurred while writing the summary.')
    else:
        summary.close()
        print()
        print('Summary written to {0}'.format(summary_path))

    return None



def download_all_metadata(drive_service):
    """
    This method scans all entries in the Google Drive account.
    All acquired metadata of each entry is stored in JSON format text files.
    """

    if not os.path.exists(RESULT_PATH):
        os.makedirs(RESULT_PATH)

    META_PATH = os.path.join(RESULT_PATH, 'Metadata')
    if not os.path.exists(META_PATH):
        os.makedirs(META_PATH)

    print('Downloading metadata...')
    try:
        results = drive_service.files().list().execute()
        items = results.get('items', [])
        for item in items:
            entry_title = item.get('title')
            print('Downloading JSON file of {0}...'.format(entry_title))

            #Request metadata of a file based on file id
            request_uri = BASE_URI + item.get('id')

            resp, content = drive_service._http.request(request_uri)
            if resp.status != 200: #Not OK
                print(resp.status)
                print('An error occurred while downloading \
                        JSON file of {0}.'.format(file_title))
                continue

            #JSON file stored in the format of [file title]__[file id].json
            json_filename = '{0}__{1}.json'.format(entry_title, item.get('id'))
            json_path = os.path.join(META_PATH, json_filename)

            json_file = open(json_path, 'wb')
            json_file.write(content)
            json_file.close()
            print('{0} JSON file written.'.format(entry_title))
    except errors.HttpError:
        print('An error occurred while downloading metadata.')
    except IOError:
        print('An error occurred while writing metadata as JSON file.')

    return None



def download_all_files(drive_service):
    """
    This method downloads all files in the Google Drive account.
    Only those files with content will be downloaded.
    """

    if not os.path.exists(RESULT_PATH):
        os.makedirs(RESULT_PATH)

    CONTENT_PATH = os.path.join(RESULT_PATH, 'Content')
    if not os.path.exists(CONTENT_PATH):
        os.makedirs(CONTENT_PATH)

    print('Downloading files...')
    try:
        results = drive_service.files().list().execute()
        items = results.get('items', [])
        for item in items:
            if not 'fileSize' in item: #If the entry has no content, skip.
                print('{0} is not a file. Skipped.'.format(item.get('title')))
                continue

            file_title = item.get('title')
            file_id = item.get('id')
            file_extention = item.get('fileExtension')

            #Request file content based on file id
            request_uri = BASE_URI + file_id + '?alt=media'

            print('Downloading {0}...'.format(file_title))
            resp, content = drive_service._http.request(request_uri)
            if resp.status != 200:
                print(resp.status)
                print('An error occurred while downloading \
                        {0}.'.format(file_title))
                continue

            #File stored in the format of
            #[file title]__[file id].[file extention]
            file_name = '{0}__{1}.{2}'.format(
                              file_title, file_id, file_extention)
            file_path = os.path.join(CONTENT_PATH, file_name)

            file_content = open(file_path, 'wb')
            file_content.write(content)
            file_content.close()
            print('---{0} has been written to local disk.'.form
```

Revisions.py

```
#Revisions.py
#Author: Shujian Yang
#University of Central Oklahoma
#Email: yang_shujian@hotmail.com

from apiclient import errors
import os

RESULT_PATH = 'Result'

BASE_URI = 'https://www.googleapis.com/drive/v2/files/'

def list_revisions(drive_service):
    """
    Return the list of revisions for each entry in Google Drive account.
    """
    try:
        results = drive_service.files().list().execute()
        items = results.get('items', [])

        rev_summary = []

        for item in items:
            entry_title = item.get('title')
            entry_id = item.get('id')

            #Folder has no revisions
            if item.get('mimeType') == 'application/vnd.google-apps.folder':
                continue

            revisions_list = get_revisions_list(drive_service, entry_id)

            if revisions_list:
                rev_info = {}
                rev_info['entry_id'] = entry_id
                rev_info['entry_title'] = entry_title
                rev_info['revision_count'] = len(revisions_list)
                rev_summary.append(rev_info)

        return rev_summary

    except errors.HttpError:
        print('An error occurred while retrieving revisions.')

    return None



def get_revisions_list(drive_service, entry_id):
    """
    Return list of a file's revisions based on file ID
    """
    try:
        revisions = drive_service.revisions().list(fileId=entry_id).execute()
        return revisions.get('items', [])
    except errors.HttpError:
        print('An error occurred while retrieving revisions.')

    return None
```

```
def download_all_revisions_metadata(drive_service, entry_id, entry_title):
    """
    Download a file's all revisions metadata based on file ID
    """
    if not os.path.exists(RESULT_PATH):
        os.makedirs(RESULT_PATH)

    REVISION_PATH = os.path.join(RESULT_PATH, 'Revisions')
    if not os.path.exists(REVISION_PATH):
        os.makedirs(REVISION_PATH)

    revisions_list = get_revisions_list(drive_service, entry_id)
    print('Downloading revision metadata of {0}...'.format(entry_title))

    try:
        for revision in revisions_list:
            revision_id = revision['id']
            request_uri = BASE_URI + entry_id + '/revisions/' + revision_id

            resp, content = drive_service._http.request(request_uri)
            if resp.status != 200:
                print(resp.status)
                print('An error occurred while downloading \
                        revision metadata of {0}.'.format(entry_title))
                continue

            #JSON file stored in the format of
            #[file title]__[file id]_[revision id].json
            json_filename = '{0}__{1}_{2}.json'.format(
                                    entry_title, entry_id, revision_id)
            json_path = os.path.join(REVISION_PATH, json_filename)
            json_file = open(json_path, 'wb')
            json_file.write(content)
            json_file.close()
        print('Revision metadata of {0} has been downloaded.'.format(entry_title))
    except errors.HttpError:
        print('An error occurred while retrieving revisions.')
    except IOError:
        print('An error occurred while writing revision content to local disk.')

    return None
```