Digital Investigation xxx (2016) 1–17

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Cloud forensics–Tool development studies & future outlook

Vassil Roussev*, Irfan Ahmed, Andres Barreto, Shane McCulley, Vivek Shanmughan

Greater New Orleans Center for Information Assurance (GNOCIA), Department of Computer Science, University of New Orleans, New Orleans, LA 70148, USA

ARTICLE INFO

Article history:
Received 2 November 2015
Received in revised form 19 April 2016
Accepted 26 May 2016
Available online xxxx

Keywords:
Cloud forensics
SaaS
Google Docs format
Cloud-native artifacts
kumodd
kumodocs
kumofs
Future forensics

ABSTRACT

In this work, we describe our experiences in developing cloud forensics tools and use them to support three main points:
First, we make the argument that cloud forensics is a *qualitatively* different problem. In the context of SaaS, it is incompatible with long-established acquisition and analysis techniques, and requires a new approach and forensic toolset. We show that client-side techniques, which are an extension of methods used over the last three decades, have inherent limitations that can *only* be overcome by working directly with the interfaces provided by cloud service providers.
Second, we present our results in building forensic tools in the form of three case studies: `kumodd`–a tool for cloud drive acquisition, `kumodocs`–a tool for *Google Docs* acquisition and analysis, and `kumofs`–a tool for remote preview and screening of cloud drive data. We show that these tools, which work with the public and private APIs of the respective services, provide new capabilities that cannot be achieved by examining client-side artifacts.
Finally, we use current IT trends, and our lessons learned, to outline the emerging new forensic landscape, and the most likely course of tool development over the next five years.

Introduction

Cloud computing is the emerging primary model for delivering information technology (IT) services to Internet-connected devices. It abstracts away the physical compute and communication infrastructure, and allows customers to *rent*, instead of *own and maintain*, as much compute capacity as needed. As per NIST's definition (Mell and Grance, 2011), there are five essential characteristics–*on-demand self service*, *broad network access*, *resource pooling*, *rapid elasticity*, and *measured service*–that distinguish the cloud service model from previous ones. The cloud IT service model, although enabled by a number of technological developments, is primarily a business concept, which changes how businesses use and interact with IT.

Importantly, it also changes how software is developed, maintained, and delivered to its customers. The traditional business model of the software industry has been *software as a product* (SaaP); that is, software is acquired like any physical product and, once the sale is complete, the owner can use it as they see fit for an unlimited period of time. The alternative–*software as a service* (SaaS)–is a subscription-based model, and was originally offered by *Application Service Providers* (ASPs) in the 1990s. Conceptually, the move from SaaP to SaaS shifts the responsibility for operating the software and its environment from the customer to the provider. Technologically, such a shift was enabled by the acceptance of the Internet as a universal means of communications (and the resulting rapid growth in

* Corresponding author.
E-mail addresses: vassil@cs.uno.edu (V. Roussev), irfan@cs.uno.edu (I. Ahmed), aebarret@my.uno.edu (A. Barreto), smcculle@my.uno.edu (S. McCulley), volipar1@my.uno.edu (V. Shanmughan).

http://dx.doi.org/10.1016/j.diin.2016.05.001
1742-2876/

network capacities), and was facilitated by the emergence of the web browser as a standardized client user interface (UI) platform. The modern version of SaaS is hosted in the public cloud infrastructure, which enabled universal and scalable SaaS deployment.

Forensics and the cloud. The traditional analytical model of digital forensics has been client-centric; the investigator works with physical evidence carriers, such as storage media or integrated compute devices (e.g., smartphones). On the client (or standalone) device it is easy to identify where the computations are performed and where the results/traces are stored. Therefore, research has focused on discovering and acquiring every little piece of log and timestamp information, and extracting every last bit of discarded data that applications and the OS may have left behind.

The introduction of *Gmail* in 2004—the first web app in mass use—demonstrated that all the technological prerequisites for mass, web-based SaaS deployments have been met. In parallel, the introduction of the first public cloud services by Amazon in 2006 enabled *any* vendor to rent scalable, server-side infrastructure and become a SaaS provider. A decade later, the transition to SaaS is moving at full speed, and the need to understand it forensically is becoming ever more critical.

NIST has led a systematic effort (NIST, 2014) to catalogue the various forensic challenges posed by cloud computing; it is an extensive top-down analysis that enumerates 65 distinct problems. Such results provide an important conceptual framework for pinpointing the most critical issues relevant to the field.

There is also the need for a *complementary* bottom-up *synthesis* effort that starts from solutions to specific cases, and becomes more general over time. Such work is experimental at heart, and allows us to both build useful tools and practices, and to gain insight into a new field. The accumulation of small, incremental successes often ends up solving problems that, at the outset, seem daunting and intractable.

Our primary focus here is on SaaS forensics as it has the fastest growth rate, and is projected to become the most common type of service (Section Future outlook); it is also the least accessible to legacy forensic tools. In particular, SaaS breaks the underlying assumption of the client-centric world that most (persistent) data is local; in a web application (the most common SaaS implementation) both code and data are delivered over the network *on demand*, and become moving forensic targets. Local data becomes ephemeral and the local disk is merely a cache—*not* the master repository it used to be. Physical acquisition, considered by many the foundation of sound forensic practice, is often completely impractical; worse, it can be ill-defined. For example, what should the "physical" image of a cloud drive—the analog of a local disk drive—look like?

It is important to come to terms with the fact that this technological shift as a *major* development for forensics. It renders much of our existing toolbox useless, and requires that we rethink and reengineer the way we do digital forensics from the ground up. In other words, this is a *qualitatively* new challenge for forensics; unlike earlier technology developments (such as the emergence of mobile devices) it cannot be addressed by relatively minor adjustments to tools and practices.

Starting with this understanding, we built several experimental tools to help us better understand the challenge, and the potential solution approach. Our main focus has been on cloud drive services, such as *Dropbox* and *Google Drive*, as these have been among the most popular ones both with consumers and with enterprises. (We avoid the term *cloud storage* as it is broader and implies the inclusion of services like Amazon's S3.) Cloud drives provide a conceptually similar interface to the local filesystem, which is ideal for drawing comparisons with filesystem forensics.

One important assumption in our development process has been that our tools have front door access to the drive account's content. This is done for several reasons, but the main one is the desire to focus on the technical aspects of what is possible under that assumption. The history of forensics shows us that the legal system will build a procedural framework and requirements around what is feasible technically (and looks reasonable to a judge). We expect that the long-term solution rests with having legally sanctioned front door access to the data. (This does not preclude other means of acquiring credentials, but we consider it a separate, out-of-scope problem.)

Tool development case studies. Our first effort was to build a (cloud) drive acquisition tool, `kumodd`,[1] which uses the service provider's API to perform a complete acquisition of a drive's content. It supports four major services and addresses two problems we identified with client-based acquisition—partial data replication on the client and revision retrieval. It partially addressed a third issue, the acquisition of cloud-native artifacts, such as Google Docs, which do not have an explicit file representation, by acquiring snapshots in standard formats, like PDF.

Our second tool, `kumodocs`, was specifically developed to work with Google Docs, which we use as a case study on how web apps store and work with such artifacts. We were able to reverse engineer the internal changelog data structure, which stores the complete editing history of the document, to the point where we can independently store and interpret it to a meaningful degree.

The third tool, `kumofs`, focuses on bridging the semantic gap between cloud artifacts and legacy file-based tools. This is accomplished by providing a filesystem interface to the cloud drive; that is, we implement a FUSE filesystem to remotely mount the drive. This makes it available for exploration, triage, and selective acquisition using standard command-line tools (like `ls` and `cp`). While the remote mount concept is old, we introduce the idea of virtual files and folders to represent aspects of the drive, such as file revisions and format conversions, that do not have direct counterparts on the local filesystem.

We also allow for *time travel*—the ability to rewind the state of the drive as of a particular time in the past, and *(time) diff*—the ability to identify all recorded activity between two date/time points. Finally, we implemented a

[1] The tool names are derived from the Japanese word for cloud *kumo* (雲).

Please cite this article in press as: Roussev V, et al., Cloud forensics—Tool development studies & future outlook, Digital Investigation (2016), http://dx.doi.org/10.1016/j.diin.2016.05.001

query interface, which allows an investigator to filter the drive data based on the rich metadata provided by the services (over 100 attributes for *Google Drive*) and show the results as a virtual folder. In effect, we bridged the semantic gap between the POSIX and drive services without losing information in the process.

Put together, we believe the three tools form an early version of a cloud-focused suite and the work has provided us with insights, which are useful to cloud forensics research and practice.

The flow of the remainder of the discussion is as follows: Section Background provides some basic background on cloud terminology; Section What makes cloud forensics different? outlines the new environment that forensic analysts face; Sections Case study: cloud drive acquisition (kumodd) through Summary and lessons learned describe the three tools outlined above—`kumodd`, `kumodocs`, and `kumofs`—as well as a summary of our experiences on the process. Finally, we discuss our outlook on the future of cloud forensics (Section Future outlook) and conclude our discussion.

Background

Cloud computing services are commonly classified into one of three canonical models—*software as a service* (SaaS), *platform as a service* (PaaS), and *infrastructure as a service* (IaaS)—and we use this split as a starting point for our discussion. This classification, although widely accepted, is somewhat simplistic. In actual deployments, the distinctions are often less clear cut, and most IT cloud solutions (and potential investigative targets) often incorporate elements of all of these.

As illustrated on Fig. 1, it is useful to break down cloud computing environments into a stack of layers (from lower to higher): *hardware*, such as storage, and networking; *virtualization*, consisting of hypervisor allowing to install virtual machines; *operating system*, installed on each virtual machine; *middleware* and runtime environment; and *application* and *data*.

In a private (cloud) deployment, the entire stack is hosted by the owner and the overall forensic picture is very similar to the case of a non-cloud IT target. Data ownership is clear as is the legal and procedural path to obtain it; indeed, the very use of the term cloud is not particularly significant to a forensic inquiry. In a public deployment, the SaaS/PaaS/IaaS classification becomes consequential as it dictates the ownership of data and service responsibilities. Fig. 1 shows the typical ownership of layers by customer and service providers on different service models. In hybrid deployments, layer ownership can be split between the customer and the provider and/or across multiple providers. Further, it can change over time as, for example, the customer may handle the base load on owned infrastructure, but burst to the public cloud to handle peak demand, or system failures.

Software as a service (SaaS)

In this model, cloud service providers own all the layers including application layer that runs the software offered as a service to customers. In other words, customer has only indirect and incomplete control (if any) over the underlying operating infrastructure and application (in the form of policies). However, since cloud service provider (CSP) manages the infrastructure (including the application), the maintenance cost on customer side is substantially reduced. Google Gmail/Docs, Microsoft 365, Salesforce, Citrix GoToMeeting, Cisco WebEx are popular examples of SaaS, which run directly from web browser without downloading and installing any software. Their desktop and smartphone versions are also available to run on client machine. The applications have varying, but limited, presence on the client machine making the client an incomplete source of evidence; therefore, investigators would need access to server-side logs to paint a complete picture. SaaS applications log extensively, especially when it comes to user-initiated events. For instance, *Google Docs* records every insert, update, and delete operation of characters performed by user along with the timestamps, which makes it possible to identify specific changes made by different users in a document (Somers, 2014). Clearly, such information is a treasure trove for a forensic analyst, and is a much more detailed and direct account of prior events than what is typically recoverable from a client device.

Platform as a service (PaaS)

In PaaS service model, customers develop their applications using software components built into middleware. *Apprenda*, *Google App Engine*, and *Heroku* are popular examples of PaaS, offering quick and cost-effective solution for developing, testing, and deploying customer applications. In this case, the cloud infrastructure hosts customer-developed applications and provides high-level services that simplify the development process. PaaS provides full

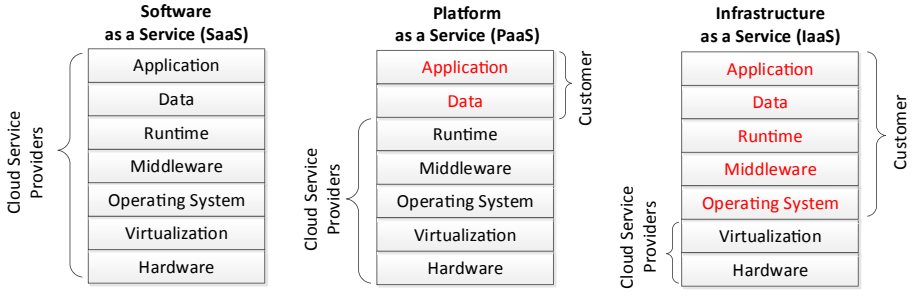


Fig. 1. Layers of cloud computing environment owned by customer and cloud service provider on three service models: IaaS, PaaS, and SaaS (public cloud).

Please cite this article in press as: Roussev V, et al., Cloud forensics—Tool development studies & future outlook, Digital Investigation (2016), http://dx.doi.org/10.1016/j.diin.2016.05.001

control to customers on the application layer including interaction of applications with dependencies (such as databases, storage etc.), and enabling customers to perform extensive logging for forensics and security purposes.

Infrastructure as a service (IaaS)

In IaaS, the CSP is the party managing the virtual machines; however, this is done in direct response to customer requests. Customers then install operating system, and applications within the machine without any interference from the service providers. Amazon Web Service (AWS), Microsoft Azure, Google Compute Engine (GCE) are popular examples of IaaS. IaaS provides capabilities to take snapshots of the disk and physical memory of virtual machines, which has significant forensic value for quick acquisition of disk and memory. Since virtual machines closely resemble physical machines, the traditional forensic tools for data acquisition and analysis can also be reused. Furthermore, virtual machine introspection provided by hypervisors enables cloud service providers to examine live memory and disk data, and to perform instant data acquisition and analysis. However, introspection is not available to customers since the functionality is supported at the hypervisor level.

In summary, we can expect SaaS and PaaS investigations to a have high dependency on logs since disk and memory image acquisition is difficult to perform due to lack of control at the middleware, operating system and lower layers. In IaaS, the costumer has control on operating system and upper layers, which makes it possible to acquire disk and memory images, and perform traditional forensic investigation.

What makes cloud forensics different?

Since our discussion is primarily focused on the technical aspects of analyzing cloud evidence (and not on legal concerns), we employ a more technical definition of digital forensics as a starting point (Roussev, 2009):

Digital forensics is the process of reconstructing the relevant sequence of events that have led to the currently observable state of a target IT system or (digital) artifacts.

The notion of relevance is inherently case-specific, and a big part of forensic analyst's expertise is the ability to identify case-relevant evidence. Frequently, a critical component of the forensic analysis is the causal attribution of event sequence to specific human actors of the system (such as users and administrators). When used in legal proceedings, the provenance, reliability, and integrity of the data used as evidence is of primary importance.

In other words, we view all efforts to perform system, or artifact, analysis after the fact as a form of forensics. This includes common activities, such as incident response and internal investigations, which almost never result in any legal actions. On balance, only a tiny fraction of forensic analyses make it to the courtroom as formal evidence.

The benefit of taking a broader view of forensic computing is that it helps us to identify closely related tools and methods that can be adapted and incorporated into forensics. In the context of cloud environments, it can help us identify the most likely sources of evidence available from a cloud service.

Forensics is a reactive technology

Digital forensics is fundamentally reactive in nature—we cannot investigate systems and artifacts that do not exist; we cannot have *best* practices before an experimental period when different technical approaches are tried, (court-)tested, and validated. This means that there is always a lag between the introduction of a piece of information technology (IT) and the time an adequate corresponding forensic capability is in place. The evolution of the IT infrastructure is driven by economics and technology; forensics merely identifies and follows the digital breadcrumbs left behind.

It follows that forensic *research* is also inherently reactive and should focus primarily on understanding and adapting to the predominant IT landscape, as opposed to trying to shape it. It is our contention that the cloud presents a new type of challenge for digital forensics, and that it requires a different set of acquisition and analysis tools. From a timing perspective, we believe that the grace period for introducing robust forensic capability for cloud environments is quickly drawing to a close, and that the inadequacies of current tools are already being felt in the field.

Ten years have elapsed since the introduction in 2006 of public cloud services by Amazon under the Amazon Web Services (AWS) brand. As of 2015, according to RightScale's *State of the Cloud Report* (RightScale, 2015), cloud adoption has become ubiquitous: 93% of businesses are at least experimenting with cloud deployments, with 82% adopting a hybrid strategy, which combines the use of multiple providers (usually in a public-private configuration). However, much of the technology transition is still ahead as 68% of enterprises have less than 20% of their application portfolio running in a cloud setup. Similarly, Gartner predicts another 2–5 years will be needed before cloud computing reaches the “plateau of productivity”, which marks mass mainstream adoption and widespread productivity gains.

Accordingly, cloud forensics is still in its infancy; despite dozens of articles in the literature over the last five years, there is a notable dearth of usable technical solutions on the analysis of cloud evidence. More importantly, we are still in a phase where the vast majority of the efforts are focused on enumerating the problems that the cloud poses to traditional forensic approachs, and looking for ways to adapt (with minimal effort) existing techniques.

The emerging forensic landscape

Most cloud forensics discussions start with the false premise that, unless the current model of digital forensic processing is *directly* and faithfully reconstructed with respect to the cloud—starting with physical acquisition—we are bound to lose all notions of completeness and integrity. The root of this misunderstanding is the use of traditional desktop-centric computational model that emerged in the

Please cite this article in press as: Roussev V, et al., Cloud forensics—Tool development studies & future outlook, Digital Investigation (2016), http://dx.doi.org/10.1016/j.diin.2016.05.001

1990s as the point of reference; the same basic approach has been incrementally adapted to work with successive generations of ever more mobile client devices. Given a successful track record of some three decades, it is entirely natural to look for ways to extend the methodology at minimum expense.

Our view is that cloud environments represent an altogether new problem space for which existing forensic approaches are woefully inadequate. For the rest of this section, we outline the main features of this new landscape.

Server-side computations are the norm. The key attribute of the client/standalone model is that practically all computations take place on the device itself. Applications are monolithic, self-contained pieces of code that have immediate access to user input and consume it instantly with (almost) no trace left behind. Since a big part of forensics is attributing the observed state of the system to user-triggered events, we (forensic researchers and tool developers) have obsessively focused on two problems—discovering every little piece of log/timestamp information, and extracting every last bit of discarded data that applications and the OS leave behind either for performance reasons, or just plain sloppiness.

The SaaS cloud breaks this model completely: the computation is split between the client and the server, with the latter doing the heavy lifting and the former performing predominantly user interaction functions. Consequently, the primary historical record of the computation is on the (cloud-hosted) server side, and not the client.

Logical acquisition is the norm. Our existing toolset is almost exclusively built to feast upon the leftovers of computations; this is becoming ever more challenging even in traditional (non-cloud) cases. For example, file carving of acquired media (Richard and Roussev, 2005) only exists because it is highly inefficient for the operating system to sanitize the media. However, for SSD devices, the opposite is true—they need to be prepared before reuse; the result—deleted data gets sanitized and there is practically no data left to carve and reconstruct (King and Vidas, 2011).

The very notion of low-level physical acquisition is reaching its expiration date even from a purely technological perspective—the current generation of high-capacity HDD (8 TB+) (Seagate) use a track shingling technique (Seagate) and have their very own ARM-based processor. The latter is tasked with identifying hot and cold data and choosing appropriate physical representation for it. The HDD device exposes an object store interface (not unlike key-value databases) that will effectively make physical acquisition, in a traditional sense, impossible. Legacy block level access will still be supported but the block identifiers and physical layout are no longer coupled as they were in prior generations of devices. By extension, the feasibility of most current data recovery efforts, such as file carving, will rapidly diminish.

Mass hardware disk encryption is another development worth mentioning, as it is becoming increasingly necessary and routine in IT procedures. This is driven both by the fact that there is no observable performance penalty, and the need to effectively sanitize ever larger and slower HDDs. The only practical solution to the latter is to use *cryptographic erase* (Kissel et al.)—always encrypt the data and dispose of the key when the disk needs to be reclaimed; practically all modern drives support this (e.g. Seagate).

In sum, the whole concept of acquiring a physical image of the storage medium is increasingly technically infeasible and is progressively less relevant as interpreting the physical image requires understanding the (proprietary) internals of the device's data structures and algorithms. The inevitable conclusion is that forensic tools will have to increasingly rely on the logical view of the data presented by the device.

Cloud storage is the norm. Logical evidence acquisition and processing will also be the norm in most cloud investigations, and it will be performed at an even higher level of abstraction via software-defined interfaces. Conceptually, the main difference between cloud computing and client-side computing is that most of the computation and, more importantly, the application logic executes on the server with the client becoming mostly a remote terminal for collecting user input (and environment information) and for displaying the results of the computation.

Logging is the default. Another consequential trend is the way cloud-based software is developed and organized. Instead of one monolithic piece of code, the application logic is almost always decomposed into several layers and modules that interact with each other over well-defined service interfaces. Once the software components and their communication are formalized, it becomes quite easy to organize extensive logging of all aspects of the system. Indeed, it becomes *necessary* to have this information just to be able to debug, test, and monitor cloud applications and services. Eventually, this will end up helping forensics tremendously as important stages of computation are routinely logged, with user input being both the single most important source of events and the least demanding to store and process.

It is a multi-version world. As an extension of the prior point, by default, most user artifacts have numerous historical versions. In Section Case study: *Google Docs* analysis (kumodocs), we will discuss a real-world illustration of this point. In contrast, our legacy acquisition, analysis, and visualization tools are designed around the concept of a single master version of an artifact. For example, there is no easy way to represent artifacts with multiple versions in current file systems; the direct mapping of each version to a named file creates a usability nightmare as it dramatically exacerbates information overload problems. In Section Case study: filesystem for cloud data (kumofs), we propose a couple of abstractions—*time travel* and *time diff*—that can alleviate these problems, but there is a clear need for much more research in this area.

Case study: cloud drive acquisition (`kumodd`)

In this section, we provide an extended summary of our experiences in building an API-based tool, `kumodd`, for cloud drive acquisition; the detailed appears in Roussev et al. (2016).

Historically, the notion of a “cloud drive” is rooted in (LAN) network drives/shares, which have been around ever since they were introduced as part of DECnet's

Please cite this article in press as: Roussev V, et al., Cloud forensics—Tool development studies & future outlook, Digital Investigation (2016), http://dx.doi.org/10.1016/j.diin.2016.05.001

implementation (DEC, 1980) in 1976. In the 1980s, filesystem-level network sharing was popularized by Sun Microsystems' *Network File System*, which later became a standard (RFC 1813). In the 1990s, the idea was extended to the WAN, and became commonly known as an *Internet drive*, which closely resembles our current concept of a cloud drive.

The main difference is that of scale—today, there are many more providers and the WAN infrastructure has much higher bandwidth capacity, which makes real-time file synchronization much more practical. Most of these services are built on top of third party IaaS offerings (such as AWS). These products have clear potential as investigative targets, and have attracted the attention of forensic researchers.

Related work: client-side analysis

The overwhelming majority of prior efforts have focused on obtaining everything from the client by employing black box differential analysis to identify artifacts stored by the agent process on the client.

Chung et al. analyzed four cloud storage services (*Amazon S3*, *Google Docs*, *Dropbox*, and *Evernote*) in search of traces left by them on the client system that can be used in criminal cases. They reported that the analyzed services may create different artifacts depending on specific features of the services, and proposed a process model for forensic investigation of cloud storage services which is based in the collection and analysis of artifacts of the analyzed cloud storage services from client systems. The procedure includes gathering volatile data from a Mac or Windows system (if available), and then retrieving data from the Internet history, log files, and directories. In mobile devices they rooted an Android phone to gather data and for iPhone they used iTunes information like backup iTunes files. The objective was to check for traces of a cloud storage service exist in the collected data.

Subsequent work by Hale (2013), analyzes the *Amazon Cloud Drive* and discusses the digital artifacts left behind after an Amazon Cloud Drive account has been accessed or manipulated from a computer. There are two possibilities to manipulate an Amazon Cloud Drive Account: one is via the web application accessible using a web browser and the other is a client application provided by Amazon and can be installed in the system. After analyzing the two methods he found artifacts of the interface on web browser history, and cache files. He also found application artifacts in the Windows registry, application installation files on default location, and an SQLite database used to keep track of pending upload/download tasks.

Quick and Choo (2013) analyzed the artifacts left behind after a *Dropbox* account has been accessed, or manipulated. Using hash analysis and keyword searches they try to determine if the client software provided by *Dropbox* has been used. This involves extracting the account username from browser history (Mozilla Firefox, Google Chrome, and Microsoft Internet Explorer), and the use of the *Dropbox* through several avenues such as directory listings, prefetch files, link files, thumbnails, registry, browser history, and memory captures. In follow-up work, Quick and Choo (2014) use a similar conceptual approach to analyze the client-side operation and artifacts of *Google Drive* and provide a starting point for investigators.

Martini and Choo (2013) have researched the operation of *ownCloud*, which is a self-hosted file synchronization and sharing solution. As such, it occupies a slightly different niche as it is much more likely for the client and server sides to be under the control of the same person/organization. They were able to recover artifacts including sync and file management metadata (logging, database and configuration data), cached files describing the files the user has stored on the client device and uploaded to the cloud environment or vise versa, and browser artifacts.

Other related work

Huber et al. (2011) focus on acquiring and analyzing the social graph of *Facebook* users by employing the official *Graph API*. The work is presented as an efficient alternative to web crawling to collect the public profile data of targeted users and their social network. To the extent that the approach uses a service API, it bears resemblance to our own approach; however, the targeted services are quite different from each other.

A number of research efforts, such as Drago et al. (2012), have focused on characterizing the network behavior and performance of cloud drive services. Although the analysis is broadly relevant, such work does not have direct forensic applications.

In sum, the only techniques that offer usable cloud drive data acquisition are client-based.

The limits of client-side analysis

The main shortcoming of client-side analysis is that it does not target the authoritative source of the data—the cloud (service). Instead, it focuses on the client-side *cache* as illustrated by Fig. 2, which sketches the typical architecture of a SaaS cloud drive.

Partial replication. The most obvious problem is that there is no guarantee that *any* of the clients attached to an account will have a complete copy of the (cloud) drive's content. *Google Drive* currently offers up to 30 TB of online storage (at $10/TB per month), whereas Amazon offers *unlimited* storage at $60/year. As data continues to accumulate online, it quickly becomes impractical to keep full replicas on all devices; indeed, with current trends, it is likely that *most* users will have *no* device with a complete copy of the data. A sound forensic solution *requires* direct access to the cloud drive's metadata to ascertain its

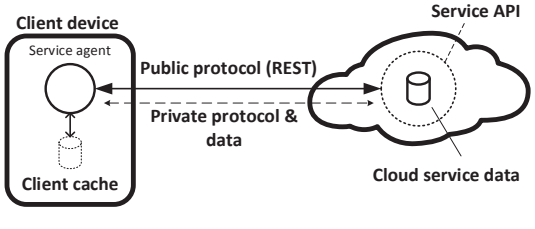


Fig. 2. SaaS cloud drive service architectural diagram.

Please cite this article in press as: Roussev V, et al., Cloud forensics—Tool development studies & future outlook, Digital Investigation (2016), http://dx.doi.org/10.1016/j.diin.2016.05.001

contents; the alternative, relying on the client cache, runs the risk of incomplete acquisition.

Revisions. Most drive services provide some form of revision history; the lookback period varies, but it is a feature users have come to expect. This is a new source of valuable forensic information that has few analogs in traditional forensic targets and investigators are not yet used to looking for it. Revisions reside in the cloud and clients rarely have anything but the most recent version in their cache; a client-side acquisition will clearly miss prior revisions; it will not even be aware of them.

Cloud-native artifacts. Courtesy of the wholesale movement to web applications, forensics needs to learn how to deal with a new problem—digital artifacts that have *no* serialized representation in the local filesystem. For example, *Google Docs* documents are stored locally as a link to the document, which can only be edited via a web app. Acquiring an opaque link is not very helpful—it is the content of the document that is of primary interest. *Google Docs* provides the means to obtain a usable snapshot of the web app artifact (PDF) but that can only be done via the service's API.

The outlined limitations of the client-side approach are *inherent* and cannot be remedied by a better implementation; therefore, we developed an approach that obtains the data directly from the cloud service.

For the rest of this section, we show how the first two concerns can be addressed by working with the public API offered by the service. After that, we will focus on the third problem, which requires the analysis of the *private* (undocumented) API and data structures used (in our case study) by *Google Docs*.

API-based acquisition

The public service API is the front door to obtaining a forensically-accurate snapshot of the content of a cloud drive, and should be adopted as a best practice. As per Fig. 2, the client component of the cloud drive (which manages the local cache) utilizes the exact same interface to perform its operations. Thus, the service API is the lowest available level of abstraction and is the appropriate interface for performing forensic acquisition. In most cases, file metadata often includes cryptographic hashes of the content, which enables strong integrity guarantee during acquisition.

The service API (and corresponding client SDKs for different languages) are officially supported by the provider and have well-defined semantics and detailed documentation; this allows for formal and precise approach to forensic tool development and testing. In contrast, black-box reverse engineering can never achieve provable perfection.

Conceptually, acquisition consists of three core phases—content discovery, target selection, and target acquisition (Fig. 3). During content discovery, the acquisition tool queries the target and obtains a list of artifacts (files) along with their metadata. In a baseline implementation this can be reduced to enumerating all available files; in a more advanced one, the tool can take advantage of search capability provided by the API (e.g., *Google Drive*) and/or perform hash-based filtering. During the selection process, the list of targeted artifacts can be filtered down by automated means, or by involving the user. The result is a (potentially prioritized) list of targets that is passed onto the tool to acquire.

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

The goal of our proof-of-concept implementation, `kumodd`, is to be a minimalistic tool for research and experimentation that can also provide a basic practical solution for real cases; we have sought to make it as simple as possible to integrate it with the existing toolset. Its basic operation is to acquire (a subset of) the content of a cloud drive and place it an appropriately structured local filesystem tree.

Kumodd

Kumodd is split into several modules across three logical layers: dispatcher, drivers, and user interface (Fig. 4). The dispatcher (`kumodd.py`) is the central component which receives parsed user requests, relays them to the appropriate driver, and sends back the result. The drivers one for each service, implement the provider-specific protocol via the web API. The tool provides two interfaces—a command-line one (CLI) and a web-based GUI.

The general format of the `kumodd` commands is:

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

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

The `[action]` argument instructs `kumodd` on what to do with the target drive: `-l` list stored files (as a plain text table); `-d` download files (subject to the `[filter]` specification); and `-csv <file>` download the files specified by the file (in CSV format). The `-p <path>` option can be used to override the default, and explicitly specify the path to which the files should be downloaded.

The `[filter]` parameter specifies the subset of files to be listed/downloaded based on file type: `all`—all files present; `doc`-all Microsoft Office/Open Office document files (`.doc/.docx/.odf`); `xls`-spreadsheet files; `ppt`-presentations files; `text`—text/source code; `pdf`—PDF files. In addition, some general groups of files can also be specified: `officedocs`—all document, spreadsheet and presentation files; `image`—all images; `audio`-all audio files; and `video`-all video files.

User authentication. All four of the services use the OAuth2 (http://oauth.net/2/) protocol to authenticate the user and to authorize access to the account. When `kumodd` is used for the first time to connect to a cloud service, the respective driver initiates the authorization process which requires the user to authenticate with the appropriate

Please cite this article in press as: Roussev V, et al., Cloud forensics—Tool development studies & future outlook, Digital Investigation (2016), http://dx.doi.org/10.1016/j.diin.2016.05.001

Fig. 3. Acquisition phases.

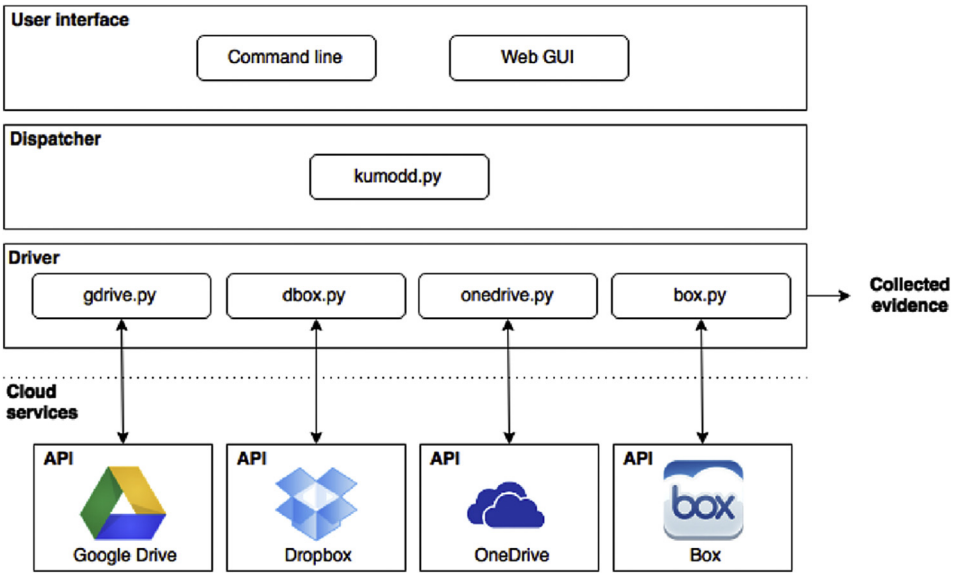


Fig. 4. *Kumodd* architectural diagram.

credentials (username/password). The tool provides the user with a URL that needs to be opened in a web browser, where the standard authentication interface for the service will request the relevant credentials.

Content discovery. The discovery is implemented by the *list* (`-l`) command, which acquires the file metadata from the drive. As with most web services, the response is in JSON format; the amount of attribute information varies widely based on the provider, and can be substantial (Google Drive). Since it is impractical to show all of it, the list command outputs an abbreviated version with the most essential information-file name, size, etc.—formatted as a plaint text table. The rest is logged as a CSV file in the `./localdata` folder with the name of the account and service. The stored output can be further processed either interactively, by using a spreadsheet program (Fig. 5), or by using Unix-style command line tools, thereby enabling subsequent selective and/or prioritized acquisition.

Acquisition. The acquisition is performed by the *download* (`-d`) command and can either be performed as a single discovery-and-acquisition step, or it can be targeted by providing a list of files (with `-csv`).

A list of successfully downloaded files is displayed with information such as download date, application version, username, file ID, remote path, download path, revisions, and cryptographic hashes. This information is logged locally, as is the original JSON metadata obtaines from the service.

Revisions. By default, `kumodd` automatically enumerates and downloads all the revisions for the files selected for acquisition; the number of available revisions can be previewed as part of the file listing (Fig. 5, column D). During download, the individual revisions' filenames are generated by prepending the revision timestamp to the base filename and can be viewed with the regular file browser, e.g.:

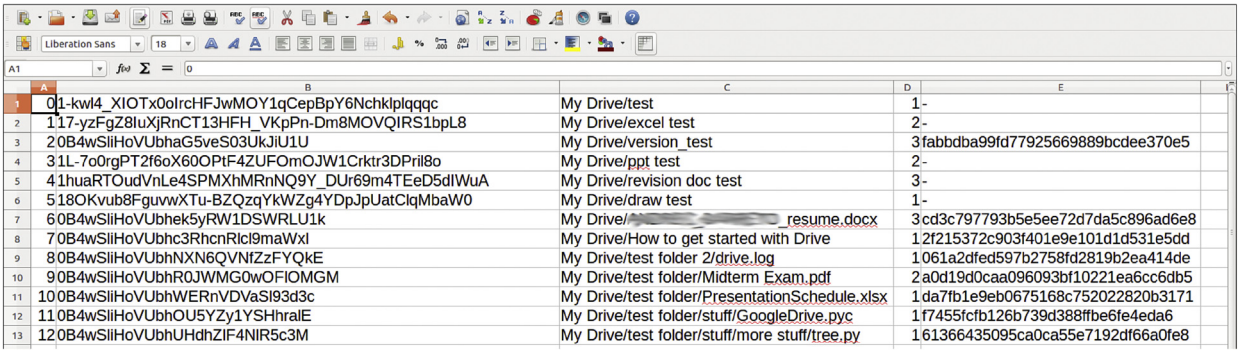
```
(2015-02-05T08:28:26.032Z) resume.docx 8.4kB
(2015-02-08T06:31:58.971Z) resume.docx 8.8kB
```

Cloud-native artifacts (Google Docs). One new challenge presented by the cloud is the emergence of *cloud-native* artifacts—data objects that have no serialized representation on the local storage, and cannot ever be acquired with client-side methods. *Google Docs* is the primary service we are concerned with in this work, however, the problem readily generalizes to many, if not most, SaaS/web applications. One of the critical differences between native applications and web apps is that the code for the latter is dynamically downloaded at run time and the persistent state of the artifacts is stored back in the cloud. Thus, the serialized form of the data (usually in JSON/XML) is an internal application protocol that is not readily renderable with a standalone application.

In the case of *Google Docs*, the local *Google Drive* cache contains only a link to the online location, which creates a problem for forensics. Fortunately, the API offers the option to produce a snapshot of the document/spreadsheet/presentation in several standard formats including text, PDF, and MS Office.

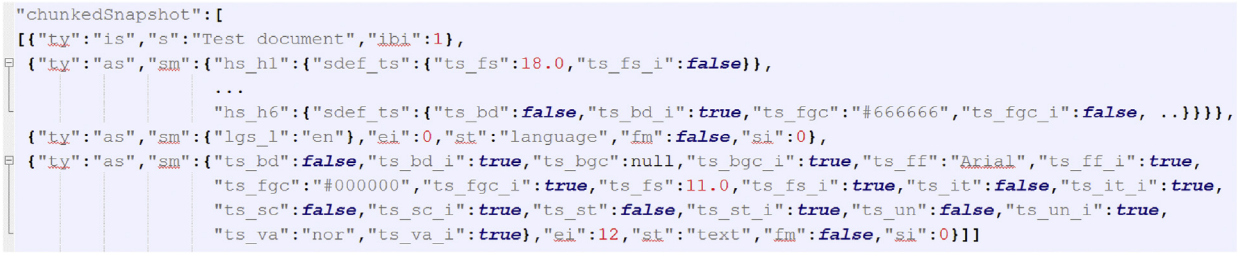
At present, `kumodd` automatically downloads a PDF snapshot of all Google Docs encountered during acquisition. Although this is clearly a better solution than merely cloning the link from the cache, there is still a loss of

Please cite this article in press as: Roussev V, et al., Cloud forensics—Tool development studies & future outlook, Digital Investigation (2016), http://dx.doi.org/10.1016/j.diin.2016.05.001



	A	B	C	D	E
1	0	1-kwl4_XIOTx0oIrcHFJwMOY1qCepBpY6Nchklplqqqc	My Drive/test	1	-
2	1	17-yzFgZ8luXjRnCT13HFH_VKpPn-Dm8MOVQIRS1bpL8	My Drive/excel test	2	-
3	2	0B4wSliHoVUbhaG5veS03UkJiU1U	My Drive/version_test	3	fabbdba99fd77925669889bcdee370e5
4	3	1L-7o0rgPT2f6oX60OPtF4ZUFOmOJW1Crktr3DPril8o	My Drive/ppt test	2	-
5	4	1huaRTOudVnLe4SPMXhMRnNQ9Y_DUr69m4TEeD5dIWuA	My Drive/revision doc test	3	-
6	5	18OKvub8FguvwXTu-BZQzqYkWZg4YDpJpUatClqMbaW0	My Drive/draw test	1	-
7	6	0B4wSliHoVUbhek5yRW1DSWRLU1k	My Drive/[redacted]_resume.docx	3	cd3c797793b5e5ee72d7da5c896ad6e8
8	7	0B4wSliHoVUbhc3RhcnRlcl9maWxl	My Drive/How to get started with Drive	1	2f215372c903f401e9e101d1d531e5dd
9	8	0B4wSliHoVUbhNXN6QVNfZzFYQkE	My Drive/test folder 2/drive.log	1	061a2dfed597b2758fd2819b2ea414de
10	9	0B4wSliHoVUbhR0JWMG0wOFlOMGM	My Drive/test folder/Midterm Exam.pdf	2	a0d19d0caa096093bf10221ea6cc6db5
11	10	0B4wSliHoVUbhWERnVDVaSl93d3c	My Drive/test folder/PresentationSchedule.xlsx	1	da7fb1e9eb0675168c752022820b3171
12	11	0B4wSliHoVUbhOU5YZy1YSHhralE	My Drive/test folder/stuff/GoogleDrive.pyc	1	f7455fcfb126b739d388ffbe6fe4eda6
13	12	0B4wSliHoVUbhUHdhZlF4NlR5c3M	My Drive/test folder/stuff/more stuff/tree.py	1	61366435095ca0ca55e7192df66a0fe8

Fig. 5. Example cloud drive metadata from *Google Docs*; CSV file (partial view). Column *B* contains the unique identifier for the file, *D* has the number of stored revisions, and *E* provides a SHA1 hash of the content.



```
"chunkedSnapshot":[
[{"ty":"is","s":"Test document","ibi":1},
 {"ty":"as","sm":{"hs_h1":{"sdef_ts":{"ts_fs":18.0,"ts_fs_i":false}},
                  ...
                  "hs_h6":{"sdef_ts":{"ts_bd":false,"ts_bd_i":true,"ts_fgc":"#666666","ts_fgc_i":false, ..}}}},
 {"ty":"as","sm":{"lgs_l":"en"},"ei":0,"st":"language","fm":false,"si":0},
 {"ty":"as","sm":{"ts_bd":false,"ts_bd_i":true,"ts_bgc":null,"ts_bgc_i":true,"ts_ff":"Arial","ts_ff_i":true,
                  "ts_fgc":"#000000","ts_fgc_i":true,"ts_fs":11.0,"ts_fs_i":true,"ts_it":false,"ts_it_i":true,
                  "ts_sc":false,"ts_sc_i":true,"ts_st":false,"ts_st_i":true,"ts_un":false,"ts_un_i":true,
                  "ts_va":"nor","ts_va_i":true},"ei":12,"st":"text","fm":false,"si":0}]]
```

Fig. 6. *Chunked snapshot* for a document containing the text "Test document" (shortened).

forensically important information as the internal artifact representation contains the complete editing history of the document.

Case study: *Google Docs* analysis (`kumodocs`)

The goal of this section is to summarize our experiences in developing an analysis tool for *Google Docs* artifacts; the detailed description is published in Roussev and McCulley (2016). We use *Google Docs* to refer to the entire suite of online office, productivity and collaboration tools offered by *Google*. We use *Documents*, *Sheets*, *Slides*, etc., to refer to specific individual applications in that suite.

Related work: DraftBack

The precursor to our work is *DraftBack* (draftback.com): a browser extension created by the writer and programmer James Somers, 2014, which can replay the complete history of a *Documents* document. The primary intent of the code is to give writers the ability to look over their own shoulder and analyze how they write. Coincidentally, this is precisely what a forensic investigator would like to be able to do—rewind to any point in the life of a document, right to the very beginning.

In addition to providing the in-browser playback (using the *Quill* open source editor (Chen and Mulligan)) of all the plaintext editing actions—either in fast-forward, or real-time mode—*DraftBack* provides an analytical interface which maps the time of editing sessions to locations in the document.

This can be used to narrow down the scope of inquiry for long-lived documents. Somers' work, although not motivated by forensics, is among the first examples of SaaS analysis that does not rely on trace data resident on the client—all results are produced solely by (partially) reverse engineering the web application's data protocol.

These observations served as a starting point of our own work, which seeks to build a true forensic tool that understands the needs of the investigative process.

Documents

In 2010, *Google* unveiled a new version of *Google Docs* (Google, 2010a), allowing for greater real-time online collaboration. The introduced new *Documents* editor, named *kix*, handles rendering elements like a traditional word processor; this is a clear break from prior practices where an editable HTML element was used. *Kix* was "designed specifically for character-by-character real time collaboration using operational transformation" (Google, 2010b). (*Operational transformation* is a concurrency management mechanism that eschewes preventive locking in favor of reactive, on-the-fly resolution of conflicting user actions (Ellis and Gibbs, 1989).)

Another important technical decision was to keep a detailed log of document revisions that allows users to go back to *any* previous version; this feature is available to any collaborator with editing privileges.

Google's approach to storing the revisions is different from most other solutions; instead of keeping a series of snapshots, the complete log of editing actions (since the creation of the document) is kept. When a specific

version is needed, the log is replayed from the beginning until the desired time; replaying the entire log yields the current version. This design means that, in effect, there is no delete operation that irrevocably destroys data, and that has important forensic implications.

To support detailed revisions, as well as collaborative editing, user actions are pushed to the server as often as every 200 ms, depending on speed of input. In collaboration mode, these fine-grained actions, primarily insertions and deletions of text, are merged on the server end, and a unified history of the document is recorded. The actions, potentially transformed, are then pushed to the other clients to ensure consistent, up-to-date views of the document.

The number of major revisions available via the public API corresponds to the major revisions shown to user in the history. Major style changes seem to prompt more of those types of revisions; for example, our working document where we kept track of our experiments has over 5100 incremental revisions but only six major one. However, the test document we used for reverse engineering purposes has 27 major revisions with less than 1000 incremental ones. The passage of time since last edit appears to play a role, but starting a new editing session does not seem to be enough to trigger a new major revision.

The internal representation of the document, as delivered to the client, is in the form of a JSON object called *changelog*. The structure is deeply nested but contains one array per revision, with most elements of the array containing *JavaScript* objects (key-value pairs). Each array ends with identifying information for that revision as follows: an epoch timestamp in Unix format, the Google ID of the author, revision number, session ID, session revision number, and the revision itself.

Each time the document is opened, a new session is generated, and the number of revisions that occur within that session are tracked. Some revisions, such as inserting an object, appear as a single entry with multiple actions in the form of a transaction. The latter contains a series of nested dictionaries; the keys in the dictionary are abbreviated (2–4 characters), but not outright obfuscated.

The changelog contains a special *chunked snapshot* object (Fig. 6), which contains all the information needed to create the document as of the starting revision. The length of the snapshot varies greatly depending on the number of embedded *kix* objects and paragraphs; it has only two entries (containing default text styles) for revisions starting at 1.

For any revision with text in the document, the first element of the snapshot consists of a plaintext string of all text in the document, followed by default styles for title, subtitle, and headings *h1* through *h6*, language of the document, and first paragraph index and paragraph styles. The next several elements are all *kix* anchors for embedded objects like comments or suggestions, followed by a listing of each contiguous format area with the styles for those sections that should be applied, as well as paragraphs and associated IDs used to jump to those sections from a table of contents.

Following the snapshot, there is an array for each revision-log entries describing the incremental changes. For example, in a document description from revision 200 to 300, there would be a snapshot of the state at revision 200, followed by 100 entries in the changelog describing each individual change from 200 to 300; some of these may be transactions with multiple simultaneous modifications. The ability to choose the range of changes to load, allows *kix* to balance the flexibility of allowing users to go back in time, and the need to be efficient, and not replay ancient document history needlessly.

The changelog for a specific version can be obtained manually by using the development tools built into the browser. After logging in and opening the document, the list of network requests contains a load URL of the form:

https://docs.google.com/documents/d/<doc_id>/load?

`<doc_id>`, where `doc_id` is the unique document identifier (Fig. 7).

To facilitate automated collection, we built a *Python* download tool that uses the Google Drive API to acquire the changelog for a given range of versions. It also parses the JSON result and converts it into a flat CSV format that is easier to process with existing tools. Each line contains a timestamp, user id, revision number, session id, session revision, action type, followed by a dictionary of key-value pairs involved in any modifications. This format is closer to that of traditional logs and is easier to examine manually, and to process with standard command-line text manipulation tools. The style modification are encoded in dictionaries so that they can be readily used (in *Python*, or *JavaScript*) to replay the events in a different editor.

The first stage in this process is to obtain the plaintext content of the documents, followed by the application of the decoded formating styles, and the addition of embedded objects (like images). Once the changelog is acquired, obtaining the plaintext is relatively easy by applying all string insert and delete operations, and ignoring everything else.

Actions that manipulating page elements, such as a table, equation, picture, etc., have a type of `ae` (add element), `de` (delete element), or `te` (tether element); the latter is associated with a kix anchor and kix id. Element insertions are accompanied by a multiset of style adjustments, containing large dictionaries of initialization values. Objects like comments and suggestions only contained anchor and id information in the changelog, and no actual text content.

Picture element insertions contain source location (URL), with uploaded files containing a local URL accessible through HTML5's FileSystem API.[2] Inserting an image from Google Drive produces a source URL in the changelog from the googleusercontent.com domain (Google's CDN) that remains accessible for some period of time. After a while (next day) the URL started reporting an http permissions error (403), stating client does not have permission. The observed behavior was readily reproducible.

Upon further examination of the HTML elements in the revision document, we established that they were referencing a different CDN link, even immediately after

[2] `filesystem:` https://docs.google.com/persistent/docs/documents/<doc_id>/image/<image_id>.

Please cite this article in press as: Roussev V, et al., Cloud forensics–Tool development studies & future outlook, Digital Investigation (2016), http://dx.doi.org/10.1016/j.diin.2016.05.001

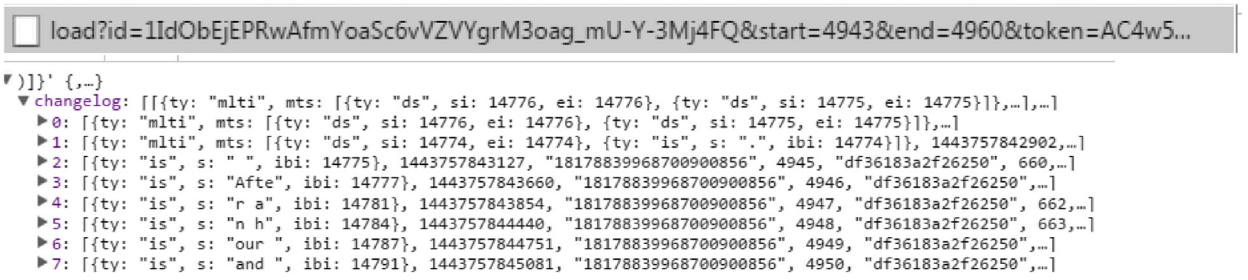


Fig. 7. Example *load* request and changelog response.

insertion. As expected, images inserted from URLs also had a copy in the CDN given that the source might not be available after insertion. One unexpected behavior was that the CDN link continued to work for several days *after* the image was deleted from the document. Further, the link was apparently public accessing it did not require being logged into a Google account, or any other form of authentication.

By analyzing the network requests, we found that the (internal) *Documents* API has a `renderdata` method. It is used with a POST request with the same headers and query strings as the `load` method used to fetch the changelog:

https://docs.google.com/document/d/<doc_id>/renderdata?id=<doc_id>

The `renderdata` request body contains, in effect, a bulk data request in the form:

```
renderOps:{"r0":["image",{"cosmoId":"1dv ... cRQ",
                          "container":"1Ss ... xps"}],
           "r1":["image",{"cosmoId":"1xv ... 3df",
                          "container":"1Ss ... xps"}],
          ...

}
```

The `cosmoId` values observed correspond to the `i_cid` attribute of embedded pictures in the changelog, and the `container` is the document id. The `renderdata` response contained a list of the CDN-hosted URLs that are world readable.

To understand the behavior of CDN-stored images, we embedded two freshly taken photos (never published on the Internet) into a new document; one of the images was embedded directly from the local file system, the other one via Google Drive. After deleting both images in the document, the CDN-hosted links continued to be available (without authentication); this was tested via a script which downloads the images every hour and those remained available for the duration of the test (72 h).

In a related experiment, we embedded two different pictures in a similar way in a new sheet. Then, we deleted the entire document from Google Drive; the picture links remained live for approximately another hour before disappearing. Taken together, the experiments suggests that an embedded image remains available from the CDN, as long as *at least one revision* of a document references it; once all references are deleted, the object is garbage collected. Forensically, this is an interesting behavior that can potentially uncover very old data, long considered destroyed by its owners.

Access to embedded Google Drawings objects is a little simpler-the changelog references them by a unique drawing id. The drawing could then be accessed by a docs.google.com URL,[3] which *does* require Google authentication and appropriate access permissions.

Case study: filesystem for cloud data (`kumofs`)

Motivation. The previous two studies focused on the basic questions of cloud data acquisition and artifact analysis. In the course of our work, we observed that cloud data objects come with a much richer set of metadata than files on the local filesystem. For example, a *Google Drive* file can have over 100 attributes that can both external (name, size, timestamps) and internal, such as image size/resolution, GPS coordinates, camera model, exposure information, and others. This creates a clear opportunity to perform much more effective triage and initial screening of the data *before* embarking on a costly acquisition. Listing 1 provides an illustrative sample (adopted from Google) of some of the *extended* attributes available, in addition to standard attributes like name, size, etc. It should be clear that some of them, like *md5Checksum*, have readily identifiable forensic use; others offer more subtle testimonials on the behavior of users, and their relationship with other users.

One (somewhat awkward) problem is that current tools are not ready to ingest and process this bounty of additional information. In particular, they are used to extracting internal metadata directly from the artifact, and are not ready to process additional external attributes that are not normally present in the filesystem. This underlines the need to develop a new generation of cloud-aware tools. In the mean time, we seek to provide a solution that allows us to utilize existing file tools on data from cloud drive services.

Listing 1. Extended *Google Drive* file attributes (sample)

[3] https://docs.google.com/drawings/d/<drawing_id>/image?w=<width>&h=<height>.

```
"md5Checksum": string,
"description": string,
"mimeType": string,
"originalFilename": string,
"fileExtension": string,
"fullFileExtension": string,
"webViewLink": string,
"webContentLink": string,
"thumbnailLink": string,
"thumbnail": {
  "image": bytes,
  "mimeType": string
},
"labels": {
  "starred": boolean,
  "trashed": boolean,
  "restricted": boolean,
  "viewed": boolean
},
"modifiedByMeDate": datetime,
"lastViewedByMeDate": datetime,
"markedViewedByMeDate": datetime,
"sharedWithMeDate": datetime,
"sharingUser": {
  "displayName": string,
  "picture": {"url": string},
},
"owners": [{
    "displayName": string,
    "picture": {"url": string},

}],
"lastModifyingUserName": string,
"lastModifyingUser": {
  "displayName": string,
  "picture": {"url": string},
},
"ownedByMe": boolean,
"editable": boolean,
"shareable": boolean,
"copyable": boolean,
"shared": boolean,
"explicitlyTrashed": boolean,
"appDataContents": boolean,
"headRevisionId": string,
"imageMediaMetadata": {
  "width": integer,
  "height": integer,
  "location": {
    "latitude": double,
    "longitude": double,
    "altitude": double
  },
    "date": string,
    "cameraMake": string,
    ...
  }
```

Filesystem access. At first glance, it may appear that this is a trivial problem-after all, cloud drives have clients that can automatically sync local content with the service. In other words, given credentials for the account, an investigator could simply install the client, connect to the account, and wait for the default synchronization mechanism to download all the data, and then can apply the traditional set of tools.

Unfortunately, this approach has a number of pain points: a) it does not allow for metadata-based screening of the data (e.g., by crypto hash); b) full synchronization could take a very long time, and the investigator would have no control over the order in which it is acquired; c) clients are designed for two-way synchronization, which makes them problematic from a forensic integrity standpoint.

We set out to design and implement a tool that addresses these concerns, and provides the following: a) read-only, POSIX-based access to all files on the cloud drive; b) means to examine the revision history of the files; c) means to acquire snapshots of cloud-native artifacts; d) query interface that allows metadata-based filtering of the artifacts, and allows selective, incremental, and prioritized acquisition of the drive content.

Design of `kumofs`. The starting point of our design is the choice of FUSE (Henk and Szeredi) as the implementation platform. FUSE is a proxy kernel driver, which implements the VFS interface and routes all POSIX system calls to a program in user space. This greatly simplifies the development process, and thousands of systems have been implemented for a wide variety of purposes. In the context of forensics, FUSE is used by `mountewf` (Metz) to provide filesystem access to EWF files; also Richard et al. (2007) and Dutch National Police Agency have proposed its use to provide an efficient filesystem interface to carving results.

Kumofs consists of five functional components: command line module, filesystem module, authentication module, cache manager, and query processor (Fig. 8). The command line module, provides the interface to all functions via the `kumofs` command. The filesystem module keeps track of all available metadata and implements all the POSIX calls; it implements multiple views of the artifacts by means of virtual files and folders (discussed below). The authentication module manages the authentication drivers for individual services, and maintains a local cash of the credentials. The cache manager maintains a prioritized queue of download requests, keeps a persistent log of all completed operations, and handles file content requests. The query processor provides the means to list, query, and filter files based on all the metadata, and to create virtual folders for the results.

Mount and unmount. To mount a cloud drive, we issue a command of the form

```
kumofs mount [service] [account] [mount-dir]
```

where `[service]` is one of the supported services (`gdrive`, `dbox`, `box`, `onedrive`), `[account]` is of the form `user@domain`, and `[mount-dir]` is the mount point. For example:

```
kumofs mount -gdrive joe@example.com gdrive/
    joe
```

The first mount for a specific account triggers *OAuth2* authentication (as with `kumodd`); after it is complete, the credentials are cached persistently. Following the authentication and app authorization, `kumofs` downloads all the file metadata and concludes the process. At this point, the

Please cite this article in press as: Roussev V, et al., Cloud forensics–Tool development studies & future outlook, Digital Investigation (2016), http://dx.doi.org/10.1016/j.diin.2016.05.001

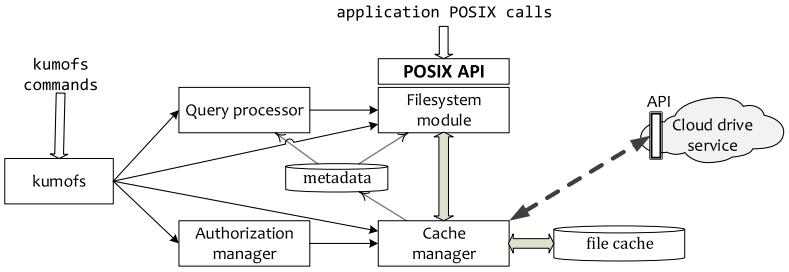


Fig. 8. `kumofs` architectural diagram.

user can navigate the files and folders using the mount point. Unmounting an account is done with

```
kumofs umount [mount-dir]
```

File download. The download of a file can be accomplished in two ways-synchronously, or asynchronously. The former blocks the requestor until the download is complete, whereas the latter adds a request to the download queue maintained by the cache manager, and returns immediately.

Synchronous download is triggered by the *fopen* system call invoked, for example, by standard commands like `cp`, and `cat`. File contents is always served from the local cache, so the download cost is paid only once regardless of how it is initiated. The cache persists across sessions, with its contents verified during the mount operation (using the crypto hashes in the metadata).

Asynchronous download is initiated with the `get` and `dl` (download) commands:

```
kumofs [get|dl] [files]
```

where standard globbing patterns can be used to specify the target `[files]`. The only difference between the get and download commands is that the former places the request at the head of the queue, whereas the latter appends it to the end. The `kumofs qstatus` command lists the state of all currently active requests; `kumofs qlog` shows the log of completed requests and their outcome (success/failure).

At any point, the analyst can choose to simply acquire the rest of the files (subject to a configuration file) with `kumofs dd`.

Virtual files & folders. Recall that a *Google Drive* account may contain *Google Docs* artifacts that have no local representation, although the API provides the means to export snapshots in different formats. By default, `kumofs` provides virtual file entries for the different export formats. For example, for a Google document called summary, the system will create file entries such as:

```
summary.gdoc.docx
summary.gdoc.odt
summary.gdoc.txt
summary.gdoc.pdf
```

It is convenient to extend the idea of virtual files to include virtual folders; it allows us to elegantly handle revisions while maintaining the filesystem abstraction. For every versioned file, such as *summary.txt*, we create a folder called *summary.txt.REVS* in which we create a file entry for each available version: *000.summary.txt*, *001.summary.txt*, …, *NNN.summary.txt* with the appropriate size and timestamps. This makes it easy to run, for example, file processing tools on successive versions. To avoid unnecessary clutter, we allow the analyst to turn the revision folders on/off with `kumofs revs [on|off]`.

We provide two views of deleted files; one is through the *.DELETED* folder in the root directory and contains the full directory structure of the removed files (*a la* recycling bin). The second one is an optional per-folder view that can be turned on/off with `kumofs del [on|off]`. While on, for every folder containing deleted files, it creates a .DELETED subfolder which enumerates them.

Time travel. One aspect of cloud forensics that analysts will have to adjust to is the abundance of time and versioning data. On a traditional filesystem, there is a single instance of the file and versioning is entirely up to the user (or some piece of middleware). As a consequence, there is no explicit representation of the system in a prior state (e.g., as of 23 days ago) that can be examined with standard tools, like a file browser.

Kumofs provides the *time travel* (`tt`) command, which sets the state of the mounted cloud drive as of a particular date/time, allowing the analyst to see the actual state of the system as of that time.[4] For example:

```
kumofs tt "Aug-31-2011 5:00p"
```

Going a step further, it is possible to save such views by creating virtual folders for them:

```
kumofs tt "Aug-31-2011 5:00p" state/Aug-31
kumofs tt "Sep-30-2011 5:00p" state/Sep-30
```

Given two (or more) snapshots, an investigator can apply a host of differential analysis techniques. We directly support this with the *time diff* command which creates a view of the filesystem (in a virtual folder) which contains all files that have been created, modified, or deleted during the period. For example, the following would yield what happened during the month of September 2011:

[4] This is based on the available version history and may not be complete.

Please cite this article in press as: Roussev V, et al., Cloud forensics–Tool development studies & future outlook, Digital Investigation (2016), http://dx.doi.org/10.1016/j.diin.2016.05.001

```
kumofs diff "Aug-31-2011 5:00p" "Sep-30-2011
    5:00p" diff/Sep
```

Metadata queries. Recall that one of the problems of remotely mounting a cloud drive is that access to much of the metadata is lost. Therefore, `kumofs` provides a supplementary means to query all of the metadata via the `mq` command:

```
kumofs mq '<filter>' show '<keys>'
```

The `<filter>` is a JSON boolean expression on the attribute values, such as `label.starred==“True”`, which would select all files marked with a star by the user. At present we support simple expressions, however, our solution is based on `jq`,[5] which has a rich query language for JSON data. With some parsing improvements we expect to have the full range of expressions.

The `show` clause is for interactive display of metadata. For example:

```
kumofs.py mq 'labels.starred=="True"' show '
    id,title,labels'
Mounting on /tmp/mq with shadow /tmp/.27oml

1D6o7PsRDIQhgGaTn5jCes50qaeFPZVVbm0gcnsQY9Ts
todo.txt
{
  "restricted": "True",
  "starred": "True",
  "viewed": "True",
  "hidden": "False",
  "trashed": "False"
}
```

As shown in the listing, our current implementation creates a temporary mount point, and places the results of the query (the selected files) there. Based on the experience, we are modifying the command to allow user-specified virtual folder to be created under the original mount point; e.g.:

```
kumofs.py mq 'labels.starred=="True"' "
    starred"
```

where *"starred"* would be the name of the virtual folder to be created under the main mount point. Moreover, executing further queries in that folder, would use as input only the files present in it; this would allow for a fluent representation of nested filtering of the results.

Summary and lessons learned

The narrative arc, which starts with evidence acquisition, continues with the analysis of cloud-native artifacts, and ends with a filesystem adaptation to allow prior tools to work with cloud data, represents the evolution of our thinking over the course of nearly a year of working with cloud services.

Some of our initial conjectures, such as the need to utilize the public API to perform the evidence acquisition, were soundly confirmed. However, our expectation that the API will solve *all* acquisition problems (and will save us the reverse engineering effort) ran aground when we approached cloud-native artifacts. These forced us to analyze the private communication protocols of web apps, in order to obtain the fine details of user actions over time.

In the case of *Google Docs*, the reverse engineering (RE) effort was not unreasonable relative to the value of the information retrieved; however, there are no guarantees that this will be the case with other services. Our preliminary examination of similar editing tools, such as *Zoho Writer* and *Dropbox Paper*, yielded both good and not-so-good news. The good news is that they generally follow Google's pattern of storing fine-grain details of every user action; the problem is that their implementations, while broadly similar, are less RE-friendly. This brings up the issue of building tool support, probably in JavaScript, to facilitate and automate the RE process.

In our `kumofs` work, we came back full circle in an effort to bring the traditional command-line toolset to the cloud. We were able to stretch the filesystem abstraction quite a bit to present a file-based view of the richer interface offered by cloud drive services. We believe that this, and similar, efforts to provide an adaptation layer that can stretch the useful lifetime of existing tools will be important in the near term. It is an open question whether this will be enough over the long term; our expectation is that we will need a much more scalable solution that will work with structured data, rather than generic file blobs.

There are a couple of lessons with respect to building cloud forensic tools that may benefit other research and development efforts.

Understanding the cloud application development process is critical. Decades of client-centric analysis has conditioned forensic researchers and practitioners to almost reflexively bring a RE-first mindset to a new problem area. However, software development practices have evolved from building monolithic standalone application to composing the functionality out of autonomous modules communicating over well-defined APIs, and distributed between clients and servers. This creates natural logging points, which developers use extensively, especially to record user input. Thus, historical information that we traditionally struggle (and often fail) to obtain is often an API call away (see *Google Docs*).

Software development is almost always easier and cheaper than reverse engineering followed by software development. As an illustration, the core of our `kumodd` prototype is less than 1600 lines of Python code (excluding the web UI) for four different services. We would expect that an experienced developer could add a good-quality driver for a new (similar) service in a day, or two, including test code.

In sum, understanding the development process and tools is critical to both the identification of relevant information sources, and the sound and efficient acquisition of the evidence data itself.

Reverse engineering is moving to the web. As illustrated by our work on *Google Docs*, critical forensic information

[5] https://github.com/stedolan/jq.

may only be acquired by reverse engineering the private web protocol and data structures. We found this flavor of RE to be easier than stepping through *x86* assembly code, or trying to understand proprietary file formats. There are a couple of reasons to think that this *may* be representative of web app RE in general. Specifically, we can readily observe the client/server communications, and the client cannot afford to have a complicated encryption/obfuscation mechanism, as it will add latency to every user interaction. We can also readily instrument the environment by injecting JavaScript code into the client context.

In sum, the nature of RE efforts is moving away from stepping through x86 assembly code, and towards reversing of network protocols and JavaScript code/data structures. The higher modularity of modern code is likely to simplify the RE effort, and to provide more historical information.

Forensics of SaaP and SaaS are (very) different. The transition from a world of software products to a world of software services requires a rethinking and of all the fundamental concepts in digital forensics. For example, in data acquisition, the old "physical is always better" principle is quickly approaching its expiration date. Intuitively, it represents the idea that we should minimize the levels of indirection between the raw source data and the investigator.

However, the insistence on obtaining the data from a physical carrier not only creates problems of access in the service world, it can lead to results that are demonstrably incomplete and potentially outright wrong. Cloud drive acquisition is an example of a scenario where the search for "physical" source leads us astray.

Instead, we propose that an investigator should always look for the most *authoritative* data source. In the case of SaaS, the authoritative source is the cloud service, while the local file system is properly viewed as a cache. We would never consider performing filesystem forensics based in the filesystem cache, yet we still look to the client cache in cloud drive acquisitions. This is clearly unsafe.

Future outlook

Attempting to predict the future is usually a thankless exercise with huge margins for error. However, digital forensics has the distinct advantage of being reactive with respect to IT developments. This gives us the time and opportunity to not so much predict forensic tool development, but to reason about what major (currently in place) IT trends mean for forensics.

The growing importance of SaaS. For the vast majority of businesses, the real benefits of cloud computing lie in divesting entirely of the need to manage the IT stack. In other words, renting VMs on AWS (IaaS) is just a first step in this direction; the end point is SaaS, which delivers (and manages) the specific end-user service the enterprise needs. According to Cisco (2014), the fraction of installed SaaS workloads on private clouds will grow from 35% in 2013 to 64% in 2018, following a 21% CAGR. At the same, IaaS is projected CAGR is only 3%, which will shrink its relative share from 49% to 22%.

The implication for forensics is that analysts will be called upon to investigate proprietary SaaS environments. Unlike IaaS cases, which are fairly similar to traditional physical deployments, such investigations will require a different set of tools and techniques.

Software frameworks & case-specific tooling. Our experience has shown us that, in the world of cloud APIs, there are some loose similarities but the specifics differ substantially, even for comparable services. Considering the *Dropbox* and *Google Drive* APIs, we see two completely different designs-the former aims for minimalism with only 18 file metadata attributes (Dropbox), whereas the latter offers well over 100 (Google) for the corresponding API response. If we add the variety of methods and calling conventions, the two services quickly diverge to the point where it is difficult to formalize a common pattern.

One part of the solution is to build an open platform that allows for the community to contribute specialized modules for different services; if successful, such an effort can be expected to cover the most popular services. Nonetheless, we can expect that forensic practitioners would need to be able to write case-specific solutions that can perform acquisition (and possibly analysis) using APIs specific to the case.

Integration with auditing. One major area of development for cloud services will be the increased level of audit logging and monitoring. Once the IT infrastructure leaves the premises, it becomes even more important to have an accurate, detailed, and trustworthy log of everything that takes place in the virtual IT enterprise. This is not just a compliance issue, but also has a direct effect on the bottom line and relationships with providers.

Since this is a common problem, we can expect common practices and standards to emerge in relatively short order. These would be a golden opportunity for forensics, and would also open the door for (low-cost) extensions to such standards to capture additional information of more specific forensic interest.

Forward cloud deployment. Looking 5–10 years ahead, we can expect that a substantial, and growing, majority of the data will be in the cloud. As it continues to accumulate at an exponential rate, it becomes increasingly costly and impractical to move a substantial part of it over the network (during acquisition) and process it elsewhere. This phenomenon is informally referred to as *data gravity*, and is openly encouraged by cloud providers as a means to retain customers.

The massive aggregation of data means that forensics will be faced with the choice of partial data acquisition (along the lines of eDiscovery), or *forward deployment* of forensic analysis tools allowing them to be collocated with the forensic target (same data center). We expect that forward deployment will become a routine part of the cloud forensic process, especially for large providers of common services. We see this as an *additional* tool that would enable fast, large-scale screening and selective acquisition.

Large-scale automation. One strong benefit of logical data acquisition is that it will enable the dramatic automation of the forensic process. Currently, the most time-consuming and least structured part of a forensic analysis is the initial acquisition, followed by low-level data

Please cite this article in press as: Roussev V, et al., Cloud forensics–Tool development studies & future outlook, Digital Investigation (2016), http://dx.doi.org/10.1016/j.diin.2016.05.001

recovery (carving) and file system analysis. In contrast, logical acquisition needs none of these and can start working with the evidence directly. Working via a logical interface implies that the structure and semantics of the data is well known and requires no reverse engineering.

In other words, forensic analysis will look a lot more like other types of data analysis, and this will bring about an unprecedented level of automation. In turn, this could bring us much closer to the proverbial "solve case" button than we currently imagine.

Conclusions

The main contributions of this work to digital forensic research and practice are as follows:

First, we made an extensive argument that cloud forensics presents a *qualitatively* new challenge for digital forensics. Specifically, web applications (SaaS) are a particularly difficult match for the existing set of forensics tools, which are almost exclusively focused on client-centric investigations, and look to local storage as the primary source of evidence.

In contrast, the SaaS model splits the processing between the client and the server component, with the server carrying most of the load. Both the client code and the data are loaded on demand over the network, and the cloud service hosts the definitive state of the user-edited artifacts; local storage is effectively a cache with ancillary functions and its contents is *not authoritative*. Therefore, the use of traditional forensic tools results in acquisition and analysis is *inherently* incomplete.

Second, we demonstrated—via a suite of tools focused on cloud drive forensics—that an API-centric approach to tool development yields forensically sound results that far exceed the type of information available by client-side analysis. This includes both complete content acquisition (containing prior revisions, and snapshot of cloud-native artifacts), and detailed analysis of *Google Docs* artifacts, which (by design) contain their full editing history. The latter was the result of extensive reverse engineering effort, which showed that web apps are likely to have much more detailed history/logs than local applications.

Third, we presented a tool, `kumofs`, which provides an adaptation layer between the local filesystem, and the cloud drive service. It provides a drive mount capability, which creates a standard POSIX interface to the content of files, thereby enabling the direct reuse of most existing tools. Kumofs also addresses the semantic gap that exists between the cloud drive service's rich versioning and metadata information, and file conversion capabilities, and the much simpler POSIX API. We introduced virtual folders, akin to database views, which allow us to show prior revisions, and to "time travel"—create views of the data as of a specific date/time. Further, they also allow us to provide a query interface that can filter the data displayed in the virtual folder based on all the metadata provided by the service.

Finally, we argued that the outline of the near-to-medium term future developments in cloud forensics can be reasonably predicted based on established IT trends. On that basis, we pointed out several developments that we expect to come to fruition. We also made the case that the field is in need of a period of active, diverse, and creative technical experimentation that will form the basis for next generation of best practices.

References

Chen J, Mulligan B, Quill rich text editor. URL https://github.com/quilljs/quill/.

Chung H, Park J, Lee S, Kang C, Digital forensic investigation of cloud storage services. 9. doi:10.1016/j.diin.2012.05.015. URL http://dx.doi.org/10.1016/j.diin.2012.05.015.

Cisco. Cisco global cloud index: forecast and methodology, 2013–2018 (updated Nov 2014). 2014. URL, http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.pdf.

DEC. DECnet DIGITAL Network Architecture (phase III). 1980. URL, http://decnet.ipv7.net/docs/dundas/aa-k179a-tk.pdf.

Drago I, Mellia M, Munafo M, Sperotto A, Sadre R, Pras A. Inside dropbox: understanding personal cloud storage services. In: Proceedings of the 2012 ACM Conference on Internet Measurement Conference, IMC'12, ACM; 2012. p. 481–94. http://dx.doi.org/10.1145/2398776.2398827.

Dropbox, Dropbox Core API file metadata. URL https://www.dropbox.com/developers-v1/core/docs#metadata-details.

Dutch National Police Agency, CarvFS user space file-system for usage with zero-storage (in-place) carving tools. URL https://github.com/DNPA/carvfs.

Ellis C, Gibbs S. Concurrency control in groupware systems. In: Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, SIGMOD'89; 1989. p. 399–407. http://dx.doi.org/10.1145/67544.66963.

Gartner's 2014 hype cycle of emerging technologies maps. URL http://www.gartner.com/newsroom/id/2819918.

Google, Google Drive REST API files resource. URL https://developers.google.com/drive/v2/reference/files.

Google. The next generation of Google Docs. 2010. URL, http://googleblog.blogspot.com/2010/04/next-generation-of-google-docs.html.

Google. Google drive blog archive: May 2010. 2010. URL, http://googledrive.blogspot.com/2010_05_01_archive.html.

Hale J, Amazon cloud drive forensic analysis. 10 (2013) 295–265. URL http://dx.doi.org/10.1016/j.diin.2013.04.006.

C. Henk, M. Szeredi, Filesystem in userspace. URL http://fuse.sourceforge.net/.

Huber M, Mulazzani M, Leithner M, Schrittwieser S, Wondracek G, Weippl E. Social snapshots: digital forensics for online social networks. In: Proceedings of the 27th Annual Computer Security Applications Conference, ACSAC'11, ACM; 2011. p. 113–22. http://dx.doi.org/10.1145/2076732.2076748.

King C, Vidas T. Empirical analysis of solid state disk data retention when used with contemporary operating systems. In: Proceedings of the 11th Annual DFRWS Conference. DFRWS'11; 2011. S111–7. http://dx.doi.org/10.1016/j.diin.2011.05.013.

Kissel R, Regenscheid A, Scholl M, Stine K, Guidelines for media sanitization, NIST Special Publication 800-88 revision 1. doi:10.6028/NIST.SP.800-88r1.

Martini B, Choo K-KR. Cloud storage forensics: owncloud as a case study. Digit Investig 2013;10(4):287–99. http://dx.doi.org/10.1016/j.diin.2013.08.005.

Mell P, Grance T. The NIST definition of cloud computing. 2011. URL, http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

Metz J, libewf wiki: mounting an ewf image. URL https://github.com/libyal/libewf/wiki/Mounting.

NIST. NIST cloud computing forensic science challenges (draft NISTIR 8006). 2014. URL, http://csrc.nist.gov/publications/drafts/nistir-8006/draft_nistir_8006.pdf.

Quick D, Choo KR, Dropbox analysis: data remnants on user machines. 10 (2013) 3–18. URL http://dx.doi.org/10.1007/978-3-642-24212-0_3.

Quick D, Choo K-KR. Google drive: forensic analysis of data remnants. J Netw Comput Appl 2014;40:179–93. http://dx.doi.org/10.1016/j.jnca.2013.09.016.

Richard G, Roussev V. Scalpel: a frugal, high performance file carver. In: Proceedings of the 2005 DFRWS Conference; 2005. URL, https://www.dfrws.org/2005/proceedings/richard_scalpel.pdf.

Richard G, Roussev V, Marziale L. In-place file carving. In: Craiger P, Shenoi S, editors. Research advances in digital forensics III. Springer; 2007. p. 217–30. http://dx.doi.org/10.1007/978-0-387-73742-3_15.

Please cite this article in press as: Roussev V, et al., Cloud forensics—Tool development studies & future outlook, Digital Investigation (2016), http://dx.doi.org/10.1016/j.diin.2016.05.001

RightScale. RightScale 2015 state of the cloud report. 2015. URL, http://assets.rightscale.com/uploads/pdfs/RightScale-2015-State-of-the-Cloud-Report.pdf.

Roussev V. Hashing and data fingerprinting in digital forensics. IEEE Secur Priv 2009;7(2):49–55. http://dx.doi.org/10.1109/MSP.2009.40.

Roussev V, McCulley S. Forensic analysis of cloud-native artifacts. In: Third Annual DFRWS Europe Conference; 2016. S104–13. http://dx.doi.org/10.1016/j.diin.2016.01.013. URL, http://www.sciencedirect.com/science/article/pii/S174228761630007X.

Roussev V, Barreto A, Ahmed I. Forensic acquisition of cloud drives. In: Peterson G, Shenoi S, editors. Research advances in digital forensics XI. Springer; 2016.

Seagate, Archive HDD data sheet. URL http://www.seagate.com/files/www-content/product-content/hdd-fam/seagate-archive-hdd/en-us/docs/archive-hdd-dS1834-3-1411us.pdf.

Seagate, Breaking capacity barriers with Seagate shingled magnetic recording. URL http://www.seagate.com/tech-insights/breaking-areal-density-barriers-with-seagate-smr-master-ti/.

Somers J. How I reverse engineered google docs to play back any document's keystrokes. 2014. URL, http://features.jsomers.net/how-i-reverse-engineered-google-docs/.

Please cite this article in press as: Roussev V, et al., Cloud forensics–Tool development studies & future outlook, Digital Investigation (2016), http://dx.doi.org/10.1016/j.diin.2016.05.001