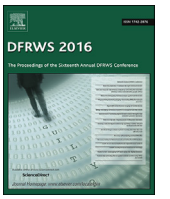
Digital Investigation 18 (2016) S55–S65

Contents lists available at ScienceDirect

Digital Investigation

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



DFRWS USA 2016 — Proceedings of the 16th Annual USA Digital Forensics Research Conference

dbling: Identifying extensions installed on encrypted web thin clients

Mike Mabey*, Adam Doupé, Ziming Zhao, Gail-Joon Ahn*

Arizona State University, 699 S Mill Ave Ste 469, Tempe, AZ, 85281, USA

Keywords:
Web thin clients
Digital forensics
Forensics on encrypted data
Chrome OS

A B S T R A C T

Researchers have developed forensic analysis techniques for so many types of digital media that there is a procedure for almost every digital media that a law enforcement officer may encounter at a crime scene. However, a new type of device has started to gain momentum in the consumer market: web thin clients. These web thin clients are characterized by native support for basic web browsing, yet other functionality relies on a combination of web applications and web storage. In fact, these devices are so different from other types of computing and storage devices that virtually all of the techniques forensic examiners and researchers typically use do not apply.

The most popular web thin client, Chrome OS, has additional forensic challenges: (1) all data associated with users is encrypted, (2) Chrome OS correctly uses TPM and Secure Boot, and (3) user data is stored on the device and in the cloud.

In this work, we present a novel approach to extract residual evidence stored on Chrome OS devices that successfully bypasses these challenges. Specifically, we are able to determine which extensions and apps are installed on an encrypted Chrome OS device, without breaking or otherwise extracting the encryption keys. Our framework, called dbling, generates signatures or fingerprints of extension and app code that persist after encryption, and we are able to use these fingerprints to identify the installed extensions and apps. We create fingerprints of 160,025 extensions for Chrome OS, we measure the uniqueness of these fingerprints, and we perform a case study by installing 14 extensions on a Chrome OS device and attempt to find their fingerprints.

© 2016 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Introduction

One of the early computing architectures was a multi-user mainframe that users connected to through terminals, or what came to be known as thin clients. However, as desktops became more powerful, this type of architecture drifted to the side, and these fat clients dominated. The industry is starting to come full circle, with the rise of cheap thin clients that leverage the power of cloud computing, the *web thin client*.

Web thin clients are a type of platform that has recently gained momentum in consumer markets to the point where they accounted for approximately 9.6% of all sales of desktops, notebooks, and tablets in 2013 (NPD Group, 2013). As of the third quarter of 2015, they also accounted for more than 51% of all sales for the K–12 market in the U.S (Swartz, 2016). Web thin clients are also poised to become "a platform for smart TVs and IoT devices" (Darvell, 2016), broadening their impact as part of the 40+ billion devices projected by ABI Research (2014) to go online by 2020.

As web thin clients continue to proliferate in the consumer marketplace, a key question of concern for the forensic community is how to extract residual evidence from web thin clients. This is a difficult question to answer for web thin clients in general, as they can store user data

* Corresponding authors.
E-mail addresses: mmabey@asu.edu (M. Mabey), doupe@asu.edu (A. Doupé), zzhao30@asu.edu (Z. Zhao), gahn@asu.edu (G.-J. Ahn).

http://dx.doi.org/10.1016/j.diin.2016.04.007
1742-2876/© 2016 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

on the device or on *any number of web and cloud service providers* due to their nature as a thin client.

We focus on a web thin client powered by Google's Chrome OS, a specialized version of the open-source Chromium OS (Chromium Projects, 2016a). Chrome OS dominates the web thin client market with the Google Chromebook (the most common form factor of Chrome OS device) and other web thin client devices. The key idea is that Chrome OS only runs a browser natively, and web applications provide all other functionality. Consequently, Chrome OS in particular offers unique forensic challenges, as (1) all data associated with users is encrypted, (2) secure boot safeguards prevent common acquisition vectors, (3) local storage is mainly a cache for online data, and (4) extensions allow users to extend the functionality of the system.

In this paper, we describe a novel approach to extract residual evidence stored on Chrome OS devices. Specifically, we are able to determine which extensions and apps are installed on an encrypted Chrome OS device *without breaking or otherwise extracting the encryption keys*, accessing the users' data stored online, or circumventing the secure boot features. We style our approach after biometrics: we generate fingerprints of extensions and app code that persist after encryption, and we are able to use these fingerprints to identify the installed extensions and apps. To accomplish this, we generate a mathematical template, which we style as a fingerprint, for the extensions that we can then compare with portions of the encrypted filesystem.

The information of installed extensions and apps is vitally important to forensic examiners when analyzing a web thin client. As the users' data may be stored on web and cloud providers, an examiner can use this information to understand *which* web and cloud providers might have the users' data. While our approach cannot reveal the data stored on the device (because we do not break the device encryption), we believe that the information of installed extensions and apps is critical information to assist a forensic examiner to discover more sources of user data and ultimately establish a more complete narrative of the incident.

The main contributions of this paper are:

- We identify important, at-rest file metadata that persists after encryption on Chrome OS devices.
- We describe a mathematical template to turn the metadata into fingerprints.
- We develop a novel approach to identify extensions and apps that are installed on Chrome OS devices using only an image of the encrypted drive, and we implement this approach in a tool called dbling.
- We evaluate our approach by creating fingerprints of 160,025 Chrome OS extensions, and we perform a case study by installing 14 extensions on a Chrome OS device and attempt to find their fingerprints.

Forensic challenges and opportunities in web thin clients

To understand our approach of extracting residual evidence stored on Chrome OS devices, we discuss the unique features of web thin clients, and particularly Chrome OS, which make forensic analysis on them both challenging and rich with opportunities.

Background: web thin clients

Thin clients are computers that rely on a server to perform the majority of their computations (United States Army, Chief Information Officer/G-6, 2013; Mell and Grance, 2011; Badger et al., 2012). A *web* thin client natively supports basic web browsing capabilities, and for any other functionality relies on a combination of web applications and web storage. Web thin clients may also support "packaged apps" that integrate with the browser to run locally to some degree, and in some cases can run offline as well.

This abstract definition of web thin clients helps identify systems of this type, but to understand them thoroughly, we must focus on a particular web thin client instance. As of this writing, there are two principal implementations on the market: Mozilla Firefox OS and Google Chrome OS. We use Chrome OS as the subject of our analysis for the following reasons:

(1) Chrome OS is built on the open-source Chromium OS, allowing us to make customizations as necessary for our experimental setup.

(2) The number of devices running Firefox OS is limited, the documentation is not as extensive as that for Chrome OS, and Firefox OS is in the process of being retargeted to connected devices (Darvell, 2016).

(3) The ecosystem of Chrome OS is more mature, has more users, and more extensions, allowing our forensic techniques to have greater impact.

To support the last point above, according to NPD Group (2013) the share of Chromebook unit sales for the U.S. commercial channel (which includes desktops, laptops, and tablets) went from 0.2% in 2012 to 9.6% in 2013. In 2014 Chromebooks then experienced a 125% growth in the U.S. business-to-business market, as shown in Table 1, and the sales of Chromebooks have continued to increase year after year (NPD Group, 2015a; 2015b; ABI Research, 2015).

Chrome OS: key features

While Chrome OS has many security features and other attributes that differ from traditional computing devices, there are four main features that make Chrome OS a challenging subject for forensic examiners. We do not discuss the other features that have no bearing on forensic analysis.

Encrypted user data

Chrome OS encrypts all user data using eCryptfs (Halcrow, 2007; Chromium Projects, 2016b, d), and it uses several keys to encrypt both the names and content of the files (see Table 2). This practice protects users' privacy from

Table 1
U.S. business-to-business (B2B) PC Category Growth.

Unit growth	2013	2014
Chromebooks	N/A	125%
Windows Notebooks	−0.4%	1.9%
Total Desktops	9.0%	5.8%
Build-To-Order (BTO) PCs (Notebooks and Desktops)	8.4%	28.3%
Thin Clients	−4.4%	27.4%

opportunistic thieves. In accordance with common practices in digital forensics, we make the assumption that the data at rest (on the hard drive) will be the main focus of an analysis. We also assume we do not have the users' credentials and that breaking the encryption is infeasible, which means the filenames and their contents are unavailable to us. However, from our analysis of eCryptfs as Chrome OS uses it, there is one feature of eCryptfs that we can use to our advantage.

As Halcrow (2007) describes, "eCryptfs is a stacked filesystem that encrypts and decrypts the files as they are written to or read from the lower filesystem." The "lower filesystem" is the set of encrypted files that are at rest on the disk, and the "upper filesystem" refers to the decrypted files. Because eCryptfs encrypts files *individually* before writing them to disk, the file's metadata remains mostly intact. Although there are slight variations in the timestamps and file size (specifically, AES is a block cipher, so eCryptfs will round the size of a file up to the nearest 4096 bytes), the metadata of the encrypted file deviates within a negligible range. This is the basis from which we create fingerprints of installed extensions and apps.

TPM and secure boot

Chrome OS stores some of the encryption keys in the Trusted Platform Module (TPM) as shown in Table 2. As described in full detail in the Chrome OS documentation (Chromium Projects, 2016d,c), a number of malicious actions will jettison the encryption keys, making the data unrecoverable.

Chrome OS also uses Secure Boot[1] to check that the firmware and kernel both pass integrity checks before booting, thereby preventing tampering with the operating system (Chromium Projects, 2016e). However, if the user has put the device in developer mode, Chrome OS skips the OS integrity check during boot, but we cannot assume that this is the case. Secure Boot also prevents an examiner from booting a custom OS for doing forensics, as has been done previously with challenging systems (Singh et al., 2009), without effectively destroying the evidence on the disk. Without being able to boot such a custom OS, it is also infeasible to perform a memory dump or memory analysis.

These features make Chrome OS more delicate than other types of systems. Any forensic techniques that aim to support Chrome OS must take this into account so as to maintain evidence integrity. Therefore, in this work we assume that we cannot extract the encryption keys stored in the TPM and that we cannot boot a custom firmware or OS.

[1] Sometimes referred to as verified boot.

Limited local storage

A benefit of web thin clients is that, due to the local hard drive being a cache of the users' cloud data, they typically use cheap, fast, and small hard drives. From a business perspective, using a 16 GB solid state hard drive provides a good user experience with quick boot times while keeping costs down. Because Chrome OS devices are thin clients and rely on servers for computation and storage, this also makes sense to only store a cache of recently used files from the user's Google Drive.

From a forensics perspective, this is a new challenge to have a suspect's device that has some *unknown subset* of their files stored locally instead of having all their data on the acquired evidence. Given the Completeness rule of evidence (Ahmad and Ruighaver, 2004), this means that by analyzing just the files on the device, an examiner cannot tell the complete story of the crime or incident.

This is why our approach for extracting the installed extensions and apps on the device is important to forensics, while we do not reveal the content of the files on the device, the installed extensions and apps points the examiner in the direction of further data. For instance, if an examiner knows that the suspect has installed the TweetDeck, Evernote, and Flickr extensions, then the examiner knows what cloud service providers to contact about extracting the suspect's data.

Extensions

Chrome supports extensions that "modify and enhance the functionality of the Chrome browser" (Chrome Developer Documentation, 2016b). In fact, in Chrome OS, extensions provide *all functionality* outside the browser itself, including the bookmarks manager, PDF viewer, and Zip file unpacker.

While extensions help Chrome OS devices mimic the familiar functionality of fat client PCs by using a variety of programs with the system, the differences here are that (1) because the data is encrypted, it is presently unclear what extra programs the suspect used with the system, and (2) it is unlikely that all the files and data created with these extensions are stored locally, both because of the caching issue discussed previously and because many of these services are web and cloud services.

Strictly speaking, there are two distinct classes of Chrome OS apps: those that run strictly as a traditional web application and store all data on the servers and those that store some data locally (for the client) to allow for "offline mode." Some rich Internet applications and HTML5 web applications, however, may act as both.

In this work we will refer almost exclusively to extensions for Chrome OS. This is not because our approach only works with extensions and not with apps. On the contrary, apps *are* extensions that include functionality not meant to be a part of the regular browser window. At the filesystem level, Chrome stores all the program files for an app in the same directory as those for extensions, and the folder is named "Extensions". Because Chrome OS treats apps as a subset of extensions, for the rest of the paper we will use the term "extensions" as a generic way of saying "extensions and apps."

Table 2
List of keys used by Chrome OS to encrypt user data. The System-Wide Key is also known as the TPM Cryptohome Key, hence the acronym. The VK consists of the file encryption key (FEK) and file name encryption key (FNEK), both are 128 bits.

Key name	Type/Bits	Scope	Origin	Encrypted by	Stored on
Storage Root Key (SRK)	RSA 8192	Hardware	Manufacturer	TPM	TPM
System-Wide Key (TPM_CHK)	RSA	System	TPM on first boot	SRK	TPM
Vault Keyset Key (VKK)	AES 256	User	Random	TPM_CHK and TK	Disk
Vault Keyset (VK)	AES 128	User	TPM on first log-in	VKK	Disk
Temporary Key (TK)	SHA 256	User	Hash of password	N/A	N/A

Disk layout

Chrome OS has a well-defined disk layout with a specific number of partitions, each with a prescribed purpose (Chromium Projects, 2016b). One partition is reserved for OEM customization, in which manufacturers may store "web pages, links, themes, etc." Another is the user state partition, or "stateful partition", in which the operating system stores all the users' encrypted files. This partition has the name `STATE` and has a mount point of `/home` once booted. The remaining partitions are reserved for the kernel, root filesystem, and partition table header and entries. This last set of partitions are not normally writable by the user, and their modification would interrupt the normal boot sequence as discussed earlier.

All models of devices running Chrome OS will have the same layout, with the possible exception of the expansion of the `STATE` partition to fill a larger hard drive. Variations on Chrome OS, including Chromium OS and CloudReady, also use this disk layout.

Because the disk layout is always consistent, our approach is compatible with almost any deployment of Chrome OS. While it is certainly possible to make alterations to the source code of Chromium OS and adjust its behavior so as to foil our approach, accounting for such a scenario is outside the scope of this work.

Anatomy of extensions

Even though it is difficult, if not impossible, to reveal data stored on the encrypted Chrome OS filesystem, extracting what extensions are installed is valuable for forensic examiners to decide next analysis steps and build a user timeline. As the goal of our project is to identify extensions installed on a Chrome OS device, we discuss relevant technical details of extensions in this section.

All extensions are installed on a per-user basis, meaning the installation is specific to each user. While there may be a set of extensions installed by default or as part of a group policy,[2] these will only be installed once the user sets up their account.

When a developer creates a new Chrome OS extension, they package it in a CRX file, which is an archive file format with a specific layout and set of custom headers (Chrome Developer Documentation, 2016a). When the browser installs the extension, it validates the headers, extracts the contents of the archive, rewrites the extension's manifest to include the developer's public key, performs localization, and converts all images to PNG format (without changing the file extension).

Chrome OS installs extensions to the following upper filesystem path:

```
/home/user/<user ID>/Extensions/<extension ID>/
                    <version>/
```

which maps to the encrypted (lower filesystem) path:

```
/home/.shadow/<user ID>/vault/user/
<encrypted Extensions>/<encrypted extension ID>/
                    <encrypted version>/
```

Chrome OS stores the extension's files according to the extension's version, so it is possible for the extension directory to have multiple versions. However, when Chrome updates an extension it removes the old version after the next restart; therefore, in practice there will only be one version directory per extension except when one was recently updated.

Extensions do not have permission to write in the folder where its files are located. This is a key insight from our analysis into the extension installation and configuration process on Chrome OS and yields two important corollaries. First, none of the user-generated or user-specific data or configurations are stored in the `Extensions` directory. Instead, extensions may store such data in one or more of the following upper filesystem paths:

```
/home/user/<user ID>/databases/
/home/user/<user ID>/IndexedDB/
/home/user/<user ID>/Local Storage/
/home/user/<user ID>/Storage/
```

Because the data stored in these locations are encrypted and have no precise structure, we cannot use them as part of an extension's fingerprint.

The second important corollary is that, for a given version of an extension, the files in the installed extension directory will be constant. These insights, coupled with the fact that file-level metadata persists after encryption, allow us to generate a fingerprint for an extension.

dbling: design and implementation

We leverage our analysis of the Chrome OS encryption scheme and the Chrome OS extension installation process

[2] See https://support.google.com/chrome/a/answer/188453.

to create a framework that is able to identify extensions installed on an encrypted Chrome OS device, without access to the encryption keys or the ability to load a custom OS. Our framework, which we call **dbling**, finds all known Chrome OS extensions, creates fingerprints of extensions, and uses the fingerprints to detect an installed extension on an encrypted Chrome OS device. Our framework consists of three phases: Enrollment, Identification, and Export. Fig. 1 depicts the three phases and their components.

Enrollment phase

Similar to the enrollment phase in biometrics, which generates a mathematical template based on the biometric trait, our approach enrolls extensions by taking a CRX file as input and producing a template of the extension.

The Enrollment phase has a crawler component and a profiler component. To identify extensions that are installed on an encrypted Chrome OS device, we must have prior knowledge of all possible extensions. Therefore, the purpose of the **Crawler** is to obtain all possible extensions, so that dbling can generate a template for each one.

The dbling crawler is a web crawler that finds all extensions currently listed on the Chrome Web Store, downloads them, and adds them to our database. Fig. 2 and Table 3 show the statistics from running the crawler over a six month period. In total we have downloaded 121,225 unique extensions. The number of versions we collected of each extension ranged from 1 to 106, as shown in Fig. 2(a). The current implementation of our crawler only downloads free extensions.

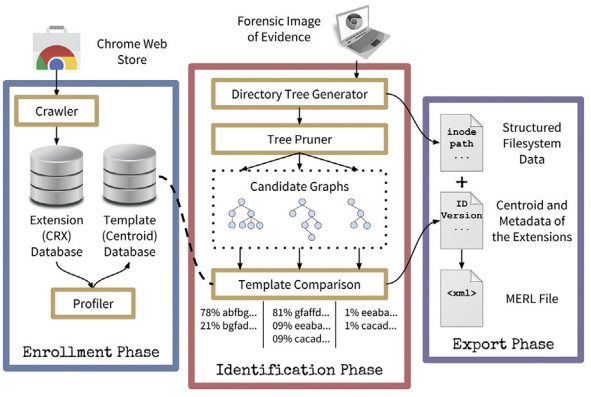


Fig. 1. The dbling framework consists of three phases: Enrollment, Identification, and Export.

The **Profiler** creates a new template for each of the new CRX files obtained by the crawler. We adapt the concept of a *centroid* to generate reliable templates for extension representations. The concept of a centroid comes from physics and represents the center of gravity of an object. Centroids have several unique characteristics that make them a good candidate for fast and reliable template generation: (C1) if two examined objects have the same centroid, they are almost certain to be the same object; (C2) when an object changes a little bit, its centroid will not change a lot; (C3) centroids are sortable, which decreases pairwise comparison time. Given these centroid characteristics, Chen et al. (2014, 2015) applied centroids to detect malicious application clones on the Android markets.

To adapt centroids for our purposes, we first transform the native CRX package into a set of unpacked directories and files. To accomplish this, we use a systematic approach that accounts for the special headers and format of CRX packages (Chrome Developer Documentation, 2016a). The process is similar to unpacking a zip archive. Then, we generate a 5D-FileTree from the unpacked files.

Definition 1 (5D-FT). A 5D-FT $= (V,A)$ is a file tree where V is the set of vertices and A is the set of arcs. In a 5D-FT each vertex has a 5D-Coordinate. A vertex in a file tree is either a directory or a file. $a(p,q) \in A$ is an arc denoting that q is a file or sub-directory in directory p.

Definition 2 (5D-Coordinate). A 5D-Coordinate is a 5-dimensional vector $\langle u,v,x,y,z \rangle$. u is the number of out-neighbor vertices that are directories, v is the number of out-neighbor vertices that are any other type of file, x is the directory or file's mode (permissions), y is the depth from the root of the file tree, and z is the file type number (block file, character device file, etc.).

To derive the 5D-FT for a version of an extension, we set the root of this tree as the version directory and walk through the set of unpacked files in it. All leaf vertices in the 5D-FT are either files or empty directories. By "files" we

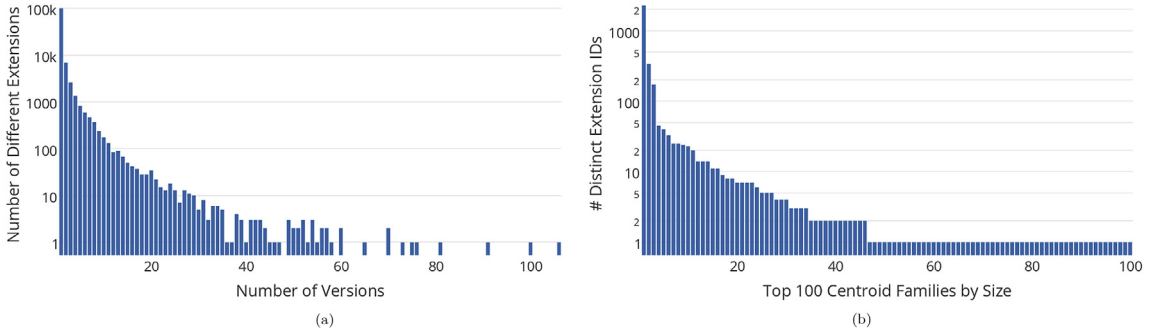


Fig. 2. More statistics from our crawler and case study. (a) Number of versions per ID. The left-most point indi-cates that 99,714 extensions have only one version in our database, the right-most point indicates one extension has 106 dierent versions. (b) Top 100 (out of 156,487) Centroid Families by size. The left-most point indicates the largest family has 2,274 extensions in it. Only 10 families have a size greater than 20. The 156,387 families not shown also have a size of 1.

Table 3
Statistics from running the crawler on the Chrome Web Store. The average new CRXs per day includes new versions of previously listed extensions as well as new extensions. We discuss the reasons why we couldn't download some CRXs in Section Technical limitations.

Crawling period	6 m/158 d
Total unique CRX IDs downloaded	121,225
Total CRXs downloaded	160,025
Total Web Store listings at start	74,253
Total Web Store listings at end	82,353
Average listing Δ per day	51.0
Average new CRXs per day	459.4
Average CRXs not downloaded per day	3.4%

mean all non-directory file types, including files of an unknown type, regular files, character and block devices, named pipes, sockets, and symbolic links. In practice, however, extension directories will only contain regular files and more directories. To generate the coordinate of a file or directory as shown in Definition 2, we execute the `stat` command on the file name of each $p \in V$ and extract the desired portions.

We define the centroid of a 5D-FT as follows:

Definition 3 (Centroid). A centroid $\vec{c}$ of a 5D-FT is a 7-dimensional vector $\langle c_u, c_v, c_x, c_y, c_z, \varpi, \phi \rangle$[3].

$$c_u = \frac{\sum_{a(p,q)\in A}(\varpi_p u_p + \varpi_q u_q)}{\varpi}, \qquad c_v = \frac{\sum_{a(p,q)\in A}(\varpi_p v_p + \varpi_q v_q)}{\varpi},$$

$$c_x = \frac{\sum_{a(p,q)\in A}(\varpi_p x_p + \varpi_q x_q)}{\varpi}, \qquad c_y = \frac{\sum_{a(p,q)\in A}(\varpi_p y_p + \varpi_q y_q)}{\varpi},$$

$$c_z = \frac{\sum_{a(p,q)\in A}(\varpi_p z_p + \varpi_q z_q)}{\varpi}, \quad \varpi = \sum_{a(p,q)\in A}(\varpi_p + \varpi_q), \quad \text{and} \quad \phi = |V|.$$

$C = \{\vec{c_0}, \vec{c_1}, \ldots, \vec{c_n}\}$ is the set of all centroids.

The ϖ_p parameter is the size of the file p, which is analogous to the weight of an object from the physics model. Because both the filesystem and the AES encryption used in eCryptfs use 4096 bytes as the block size, the encrypted file size is the original file size rounded up to the nearest 4096 bytes. To calculate ϖ_p, we compute the unencrypted file's size when encrypted and divide it by 4096 to get the number of blocks occupied by the corresponding encrypted file. Not only does this simplify our view of the file size for the context of working with eCryptfs, but it also reduces variance in the centroids of each file.

The inclusion of ϕ in the centroid was a direct result of our observations while experimenting with our approach. In our database, ϕ ranges from 3 to 36,743.

Example: Fig. 3 shows a 5D-FT example. Vertex A is the version directory. It has two child directories and no other file. Its mode is 764, which means that owner can read, write and execute, group can read and write, and others can only read. The depth from A to the version directory is 0, and it is a directory, which is type 2 in ext filesystems. So its coordinate is ⟨2,0,764,0,2⟩. Accordingly, we can calculate the centroid of this tree with the coordinates of all its vertices.

Identification phase

In our framework, the Identification phase is the part of our approach that forensic examiners will use. Because of this, the phase must address the three main steps of the forensic process: acquisition, authentication, and analysis.

The methods of acquisition (making a forensic copy) and authentication (ensuring the copy's integrity) on a Chrome OS device depend on what hardware the manufacturer used for the device. With increasing frequency, Chrome OS devices come with embedded hard drives, which make the initial acquisition difficult.[4] Assuming that the device has a removable hard drive, typical acquisition and authentication methods that are compatible with Linux operating systems will work on a Chrome OS device.

For the analysis step, our high-level goal is to identify installed extensions. However, we must first identify the extension directories in the encrypted filesystem, for which we do not have the unencrypted names of files or directories or their contents. To overcome this initial challenge, we search the encrypted filesystem to identify the **candidate graphs** that represent the most likely directories to contain user-installed extensions, generate templates for the candidate graphs, and perform a one-to-many search to determine what extensions each may be. Intuitively, we cannot perform a one-to-one verification because the files are encrypted and cannot self-identify as any particular extension.

To begin, we assume the examiner has already completed the process of acquiring and authenticating a forensic image from the evidence. The image does not need to be of the entire disk, but can be of the partition named `STATE` (Chromium Projects, 2016b), whose mount point is `/home` and stores all the users' data.[5]

The next step in the process is to create a graph of the filesystem on the disk image. To do this, we mount the image using a read-only loop device, then we walk through the filesystem, creating a vertex for each inode and an arc between parent directories and the files they contain, as described earlier.

We create a graph representing the whole filesystem to make sure we do not miss any relevant files. However, given that we are only interested in a subset of these files, we can leverage a number of identifying characteristics and label vertices we know cannot be part of an extension. With these labels, we can prune irrelevant files from the graph. In addition to the details presented in Section Anatomy of extensions, these helpful characteristics are:

(1) All directories outside the scope of `/home/.shadow` will *never* contain user-installed extension files.

(2) The `Extensions` directory will *always* be six directories deep from `/`, level α in Fig. 4.

(3) The `Extensions` directory will *always* be encrypted.

(4) The `Extensions` directory will *always* contain *only* directories (i.e. no regular files), one for each installed extension.

[3] The glyph ϖ is called "variant pi" or "pomega."

[4] We believe that this is an interesting challenge and are researching some possible methods to overcome this difficulty.

[5] This is why Fig. 4 shows `/home` as the root vertex.

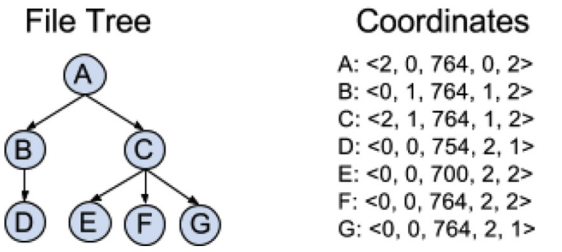


Fig. 3. A 5D-File Tree example and the 5D-Coordinates for its vertices.

(5) The directory of a specific extension will *always* contain *only* directory files at level γ, one for each version of the extension currently installed (typically 1).

(6) The extension version directory *must* be non-empty since extensions always require a manifest file.

(7) The extension version directory will *only* contain directories and regular files, no symbolic links or other types of files.

While generating the directory tree, we also use the `stat` program on each file to get its metadata, then we generate the 5D-Coordinate for the file and store it in connection with the vertex corresponding to the file. Once the graph is complete, we prune the tree by filtering out all unnecessary files using the labels we created earlier.

Having created the pruned directory tree, we next separate the candidate graphs from the directory tree as shown in Fig. 4, making them proper 5D-FTs. We then calculate the centroid for each candidate graph using the formula in Definition 2.

With the centroids of each of the candidate graphs, the next step is to perform the one-to-many matching for each of them. The naïve approach would be to attempt to match a centroid against all of the known templates. However, this would be inefficient as the size of the known templates grows. Therefore, we take advantage of two aspects of our templates. First, because centroids are vectors, we can sort them and then set a distance threshold to limit how many neighboring centroids we will include in the pairwise comparison. Second, the value of ϕ must match exactly, reducing the number of entries to consider.

Definition 4 (Centroid Family). A **centroid family** $CF_i \subseteq C$ is a set of centroids such that $\vec{c_j} = \vec{c_k}\ \forall \vec{c_j}, \vec{c_k} \in CF_i$.

However, due to the coarse granularity of our centroids,[6] it can still be the case that centroids from multiple extensions are the same value, either because they are different versions of the same extension or because the structure and metadata values are the same. We call these *centroid families*.

In essence, centroid families provide a measure for the uniqueness of the templates that we generate. Also, it is natural that multiple extensions will correspond to the same centroid. Consider the case of a simple extension that has only one file of less than 4096 bytes. In this case, all these simple extensions will have the same centroid and be in the same centroid family. Thus, when a candidate graph template matches a template of this centroid family, we cannot say for certain which extension is installed, only that it is one of the extensions in the centroid family.

Table 4 describes statistics from our centroid families. Of particular note is that 156,441 centroids correspond to unique extensions, which means that only 46 centroids correspond to multiple extensions. Fig. 2(b) investigates the distribution of centroid families to multiple extensions. In the worst case (the left-most bar in Fig. 2(b)), a single centroid corresponds to 2,275 extensions. Still, even in the worst case, this narrows down the possible extensions installed on the Chrome OS device from the 160,025 possible extensions to just 2,275.

With the potential matches for each candidate graph identified, the next step is to normalize the centroid vectors. In our experiments, we noticed that the range of values for each field in the centroid vectors is not the same. For example, the valid values for a file's 5D-Coordinate for x (the file's mode) have a range of 0–777, but z (the file type number) has a range of 1–7. To accommodate this characteristic and avoid allowing certain fields to dominate the distance calculation in Formula (1), we normalize centroids using the following definitions:

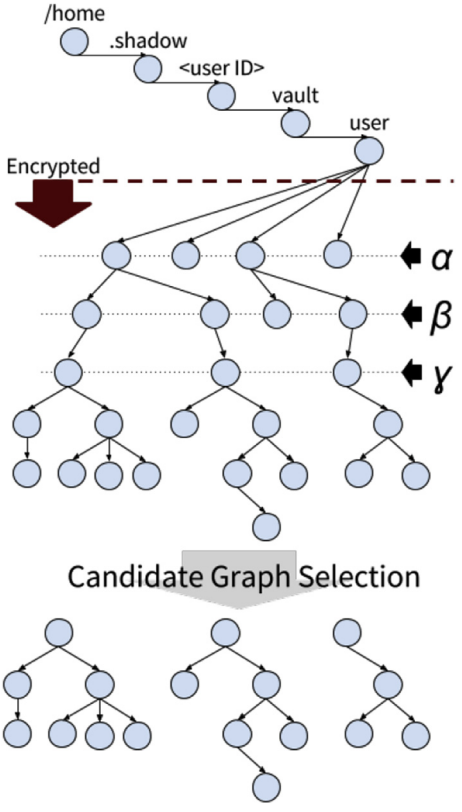


Fig. 4. Pruning process from the full directory tree to only a few candidate graphs. In the figure, the vertices at level α are the potential "Extension" directories, vertices at level β are potential extension ID directories, and vertices at level γ are potential extension version directories.

[6] This coarseness is due to the effect of encryption on the file metadata: we do not have filenames or exact file sizes.

Table 4
Statistics on centroid families. Membership numbers are for distinct extension IDs.

Total extension entries	160,025
Total centroid families	156,487
Biggest centroid family	2274
# of families >= 20	11
# of families == 1	156,441

Definition 5 (Normalized Centroid). A Normalized Centroid $\overrightarrow{c'}$ is a 7-dimensional vector, where each component c'_i (except for ϕ) is calculated as $c'_i = \frac{c_i}{cn_i}$, where c_i is the i^{th} component of the non-normalized centroid $\overrightarrow{c}$ and cn_i is the i^{th} component of the Normalizing Vector. The ϕ component is never normalized.

Definition 6 (Normalizing Vector). The Normalizing Vector is a 6-dimensional vector $\overrightarrow{cn} = \langle cn_u, cn_v, cn_x, cn_y, cn_z, cn_\varpi \rangle$ where each component cn_i of $\overrightarrow{cn}$ is the maximum value for that component in the database of collected centroids.

With the normalized centroids calculated, we can finalize the Identification phase by ranking the matches according to our confidence level in the matching. For a candidate graph a and extension x, we calculate the confidence level $conf(a,x)$ as follows:

$$conf(a,x) = \frac{e^{-\delta\left|\overrightarrow{c'_a} - \overrightarrow{c'_x}\right|}}{n} \quad (1)$$

Confidence levels will be in the interval (0,1], indicating a percentage likelihood that candidate graph a is extension x. In the above formula, δ is a weight for adjusting the rate at which our confidence drops as the distance between $\overrightarrow{c'_a}$ and $\overrightarrow{c'_x}$ increases, and $n = |CF_i|$ given $\overrightarrow{c_x} \in CF_i$.

Export phase

In the Export phase, we store the matching and ranking results from the Identification phase in a semi-structured format to allow examiners to share them. This phase of our approach is important considering the increasing emphasis from the security community to standardize intelligence formats for machine-to-machine communication and other sharing purposes (Haass et al., 2015).

To export the ranked results so as to not lose their meaning in the context of the original forensic image, we must make a connection between the evidence artifacts and the results in a cohesive format. To accomplish this, we developed an XML schema that directly references the inode information from the forensic disk image and adds the ranking information. We call these **Matching Extension Ranking List (MERL)** files. Fig. 5 shows the MERL file that lists the matching extensions with confidence levels.

Case study

We conducted a case study of dbling in a testing environment to demonstrate its effectiveness. The target device was an Acer C720 Chromebook, which we selected based on its removable NGFF hard drive. The version of Chrome OS running on the device at the time was 48.0.2564.92.

```
<merl>
  <source>
    <image_filename>chromebook_test01.img</
        image_filename>
    <mount_point>/mnt/dbling/</mount_point>
  </source>
  <creator>
    <program>dbling_merl</program>
    <version>0.0.1</version>
    <execution_environment>
      <command_line>sudo dbling_merl -v -m /mnt/dbling/<
          /command_line>
    </execution_environment>
  </creator>
  <match>
    <inode>259696</inode>
    <candidate>
      <ext_id>abfbgiabldngkapfilcncjplidclalae</ext_id>
      <ext_ver>1.0.3</ext_ver>
      <ext_name>Mangekyou1999</ext_name>
      <ext_vendor>Mangekyou1999</ext_vendor>
      <confidence>0.78</confidence>
    </candidate>
    <candidate>
      <ext_id>bgfadgcdohfieimbjkkficoionibpipa</ext_id>
      <ext_ver>2.0.1</ext_ver>
      <ext_name>ExtBoostAdopt</ext_name>
      <ext_vendor>sabrina.webdev</ext_vendor>
      <confidence>0.21</confidence>
    </candidate>
  </match>
</merl>
```

Fig. 5. A MERL file with the extensions that might match the extension version directory from the source image.

We chose 14 extensions from the "Popular Apps" category and those from a combination of the "Productivity" category that were listed as working offline. We used these criteria to help simulate the scenario where the user has installed extensions that use online services that store files somewhere in the cloud, as well as the case where the extension does more work locally by having more code installed on the Chromebook.

We also constrained the case study to extensions and versions that our crawler had downloaded. This was necessary because of the types of extensions we were unable to download (see Section Technical limitations).

With the selected extensions that met our criteria, we installed all 14 of them on the target device, took a forensic image of the disk using `dd`, then ran dbling on the image. From the MERL results, we searched for the IDs of the installed extensions and recorded their ranks among the matching centroids to get an idea for how accurately dbling was able to identify the extension. Table 5 summarizes the results from these experiments.

The τ column shows how many extensions have the same value of ϕ (number of files and directories) in the database. In this case we consider different versions of an extension separately. As the table shows, the higher the value of τ, the more difficult it was to find the correct extension. This makes sense because we are using ϕ as a pre-filter when looking for matching centroids, so the lower the value of τ the fewer centroid distances we have to calculate and rank.

From Table 5 we can see that there seems to be an inverse relation between ϕ and τ. Considering that many extensions in the Chrome Web Store are simply links to a web application, this makes sense that there are so many extensions with ϕ=5, for example.

The results from our case study show that, for complex applications, we are able to narrow down the scope of digital

Table 5
Rankings for matching the candidate graphs to the correct extension using centroids. Numbers given in ranges indicate multiple versions of the extension matched the candidate.

Extension name	Version	Offline	# Users	ϕ	τ	Rank
Lucidchart Diagrams — Desktop	1.116	Y	389,087	3507	3	1–3
Marxico	1.6.1	Y	29,379	711	7	1–7
Piconion Photo Editor	1.9.0.0	Y	24,684	370	12	2
Reditr — The Best Reddit Client	0.3.3.1	Y	53,687	134	71	1
Google Hangouts	2016.120.1336.1	N	5,468,622	150	78	1–19
Advanced REST client	4.6.0	N	1,040,107	131	102	8–12
Evernote Web	1.0.8	N	3,638,599	41	580	308
TweetDeck by Twitter	3.10	N	1,584,975	15	2808	894; 1904
Any.do	3.1.1	Y	260,355	13	3894	3217
ShiftEdit	1.43	Y	159,170	6	5766	2175
Outlook.com	1.0.2	N	1,485,767	7	9259	4318; 4520
Little Alchemy	1.4.0	Y	1,518,447	5	11,768	10,328
Google Play	3.1	N	3,624,424	5	11,768	6919; 6989
Netflix	1.0.0.4	N	1,769,793	5	11,768	9630

evidence for a forensic examiner. In the worst case, which is in fact ϕ=5, we narrowed down the scope to 11,768 potential extensions from the total of 160,025. In other cases dbling was able to identify the extension exactly.

Limitations

The limitations of our approach fall into two main categories: forensic and technical.

Forensic limitations

In our assumptions we stated that our current approach only works on Chrome OS devices that use removable hard drives. The forensic community is very familiar with the process of removing a hard drive from a device, imaging it, and analyzing the forensic image for evidence. Less straightforward is the process of acquiring a forensic copy of a solid state drive soldered directly to the motherboard, such as eMMC drives. Although there exist techniques for accomplishing this, the current cost of the specialized equipment, software, and training required is often prohibitively high. Because of this, we tested our approach on those devices for which we could leverage common practices.

Assuming that, either in our future work or by other researchers, an affordable acquisition method for these types of embedded solid state drives comes to light, the rest of our approach will still apply.

Technical limitations

Although our approach may introduce high-level principles that could apply to other encrypted filesystems, dbling cannot presently operate on anything other than the discussed web thin client operating systems because of its dependency on the disk layout discussed in Section Disk layout and the precise location of static extension files as discussed in Section Anatomy of extensions.

The success of our approach relies on a complete enrollment of all extensions, as we will never be able to detect an extension that is not enrolled. However, in practice we encountered several challenges to enrolling all possible extensions.

Chrome seems to have some interesting mechanisms in place for interacting with the Chrome Web Store that we have not yet been able to reverse-engineer. For instance, developers can opt to list their extension as being compatible only with Chrome OS, thereby disallowing installations of that extension on the Chrome browser in an arbitrary operating system.

To investigate this feature, we first attempted to get around this check by changing the “User-Agent” header of our requests to mimic Chrome OS, however we were still not able to download the extension. Next, we browsed the Chrome Web Store with the Firefox browser. Many page elements were missing and it was clear Firefox did not have all of the JavaScript files connected with the site.

These two results indicate that Chrome and Chrome OS have “insider knowledge” of how to interact with the Chrome Web Store. The evaluation of compatibility is clearly not done server-side, and the client-side browser needs to know how to fill in the gaps from the Store's HTTP responses. Since we aren't privy to these details, we were unable to spoof the identity of our downloader to obtain these extensions.

Other reasons why we were unable to download certain extensions are: (1) either the developer or Google removed the extension from the Chrome Web Store, (2) the developer released the extension as a closed beta version or by invite only (Chrome Developer Documentation, 2016b), (3) the extension is non-free.

Although our approach helps examiners identify which extensions the user has likely *installed*, dbling cannot in its current form give any indication as to which extensions the user actually *uses* or with what frequency. This feature would be a logical evolution of dbling's current abilities.

problem of extensions generating the same centroid and belonging to the same centroid family impacts our detection results. We believe that this large number may be due to published extensions that are clones of either other extensions on the Chrome OS store or very simple extensions. As shown in our case study, the more complex an extension, in both file size and number of files, the more likely it is to be unique. Furthermore, we argue that it is a

natural result of the loss of data and metadata from the encryption of the filesystem.

Related work

Garfinkel (2009) first introduced DFXML to the forensics community. We attempted to incorporate DFXML into our approach by referencing `fileobject` nodes from the MERL files instead of the inode number. However, as noted by Garfinkel et al. (2012) (and later reiterated by Nelson (2012)), a tool cannot assume "a unique path for each object in [the DFXML of] a `file system`". We learned this the hard way when, after creating the 5D-FT from the DFXML, the duplicate entries caused all sorts of issues. We finally concluded that walking the filesystem ourselves was a more straightforward approach than attempting to sanitize the DFXML. Hopefully in the future we can overcome this issue.

Chrome OS uses `ext4` as the underlying filesystem, which is also adopted by the Android and Tizen operating systems as well. Fairbanks (2012) and Fairbanks et al. (2010) presented thorough analyses of `ext4` in general for digital forensics. Kim et al. (2012) studied how to use the `ext4` filesystem journal logs to extract what files users accessed and how to recover deleted files. Eo et al. (2015) examined how to recover deleted files in Tizen. Corbin (2014) studied performing forensic analyses on Chromebooks, in which they assumed the examiner is able to log in to the device before trying to run some simple commands on the system, like `ls`, `lsof`, `grep`, etc. However, all the aforementioned approaches assume the filesystem is unencrypted, either because it is stored in plaintext (Kim et al., 2012; Eo et al., 2015) or decrypted before analysis (Corbin, 2014). This assumption dramatically limits forensic analysts' ability when they can only acquire images of encrypted filesystems.

Other researchers have studied the impact of full disk encryption on digital forensics (Casey and Stellatos, 2008). Efforts that tried to do forensics on encrypted systems all explored ways to recover the encryption keys from memory or other places (Altheide et al., 2008; Halderman et al., 2009). In these approaches, the success of forensic analysis totally relies on the success of extracting the keys. Little literature exists on directly doing forensics on the encrypted filesystems of Chrome OS or any other type of web thin clients.

To overcome the obstacles inherent to encrypted filesystems, our work follows the line of metadata analysis for digital forensics. Buchholz and Spafford (2004) discussed not only what types of information are available by analyzing metadata in FAT and NTFS, they also outlined what new types of metadata would aid forensic investigations. Olivier (2009) leveraged metadata analysis to perform forensics on databases. Castiglione et al. (2007) used document metadata to retrieve forensic information from Microsoft Office documents. Thorpe et al. (2012) examined cloud log metadata for forensic analysis of virtual machines and operating systems running in clouds.

Conclusion

It is clear that web thin clients are an exciting and expanding computing platform. They offer significant benefits for users, allowing them to leverage the power of cloud computing in an inexpensive device. However, this new class of devices brings interesting and unique forensic challenges, particularly in the case of Chrome OS, where the filesystem is encrypted by default.

By using file metadata that is preserved after encryption, we were able to extract forensic evidence without breaking the encryption or loading a custom operating system. Our approach significantly helped narrow down analysis space on the forensic image, which will eventually support both logical and sparse acquisitions. Furthermore, we believe that this is a significant first step toward comprehensive web thin client forensics.

As part of our ongoing efforts to improve dbling, we are working on a suite of tools for large-scale evaluation of our approach on all of the extensions in the Chrome Web Store. This includes trying other techniques (e.g. using SSL MITM) to understand the request format for downloading extensions so we can obtain those that were previously inaccessible.

Acknowledgments

We are grateful to Alex Nelson for his useful comments while this work was in progress. We also thank the anonymous reviewers whose comments and suggestions have significantly improved the paper. This work was supported in part by grants from the U.S. Department of Defense Information Assurance Scholarship Program and the Center for Cybersecurity and Digital Forensics at Arizona State University. The information reported here does not reflect the position or the policy of the funding agency or project sponsor.

References

ABI Research. The internet of things will drive wireless connected devices to 40.9 billion in 2020. Aug 2014. URL, https://www.abiresearch.com/press/the-internet-of-things-will-drive-wireless-connect/.

ABI Research. ABI research forecasts Chromebooks to top 8 million unit shipments and lead 2015 global growth in notebook PCs. Dec 2015. URL, https://www.abiresearch.com/press/abi-research-forecasts-chromebooks-top-8-million-u/.

Ahmad A, Ruighaver AB. Towards identifying criteria for the evidential weight of system event logs. In: 2nd Australian Computer Network & Information Forensics Conference, November 25th 2004, Perth, Western Australia, Forensic Computing – Evidence on the move from Desktops to Networks, Conference proceedings. Western Australia: School of Computer and Information Science, Edith Cowan University; 2004. p. 40–7.

Altheide C, Merloni C, Zanero S. A methodology for the repeatable forensic analysis of encrypted drives. In: Proceedings of the 1st European workshop on system security. EUROSEC '08. New York, NY, USA: ACM; 2008. p. 22–6.

Badger L, Grance T, Patt-Corner R, Voas J. Cloud computing synopsis and recommendations recommendations of the National Institute of Standards and Technology. NIST Special Publication; 2012. 800–146 URL, http://csrc.nist.gov/publications/nistpubs/800-146/sp800-146.pdf.

Buchholz F, Spafford E. On the role of file system metadata in digital forensics. Digit Investig 2004;1(4):298–309.

Casey E, Stellatos GJ. The impact of full disk encryption on digital forensics. SIGOPS Oper Syst Rev Apr. 2008;42(3):93–8.

Castiglione A, De Santis A, Soriente C. Taking advantages of a disadvantage: digital forensics and steganography using document metadata. J Syst Softw 2007;80(5):750–64.

Chen K, Liu P, Zhang Y. Achieving accuracy and scalability simultaneously in detecting application clones on Android markets. In: Proceedings of the 36th International Conference on Software Engineering. ICSE 2014. New York, NY, USA: ACM; 2014. p. 175–86.

Chen K, Wang P, Lee Y, Wang X, Zhang N, Huang H, Zou W, Liu P. Finding unknown malice in 10 seconds: mass vetting for new threats at the Google-Play scale. In: 24th USENIX Security Symposium (USENIX Security 15). USENIX Association; Aug 2015. p. 659–74.

Chrome Developer Documentation. CRX package format. 2016. https://developer.chrome.com/extensions/crx [accessed 08.04.16].

Chrome Developer Documentation. What are extensions?. 2016. https://developer.chrome.com/extensions [accessed 08.04.16].

Chromium Projects. Chromium OS FAQ: What's the difference between Chromium OS and Google Chrome OS?. 2016. URL, https://www.chromium.org/chromium-os/chromium-os-faq#TOC-What-s-the-difference-between-Chromium-OS-and-Google-Chrome-OS- [accessed 08.04.16].

Chromium Projects. Disk format. 2016. URL, https://www.chromium.org/chromium-os/chromiumos-design-docs/disk-format [accessed 08.04.16].

Chromium Projects. Firmware verified boot crypto specification. URL. https://www.chromium.org/chromium-os/chromiumos-design-docs/verified-boot-crypto; 2016c [accessed 08.04.16].

Chromium Projects. Protecting cached user data. 2016. URL, https://www.chromium.org/chromium-os/chromiumos-design-docs/protecting-cached-user-data [accessed 08.04.16].

Chromium Projects. Verified boot. 2016. URL, https://www.chromium.org/chromium-os/chromiumos-design-docs/verified-boot [accessed 08.04.16].

Corbin G. The Google Chrome operating system forensic artifacts. 2014. Capstone report, Utica College.

Darvell J. Firefox OS. Jan 2016. URL, http://www.linuxjournal.com/content/firefox-os.

Eo S, Jo W, Lee S, Shon T. A phase of deleted file recovery for digital forensics research in Tizen. In: IT Convergence and Security (ICITCS), 2015 5th International Conference on. IEEE; 2015. p. 1–3.

Fairbanks KD. In: An analysis of ext4 for digital forensics. Digit Investigation 9, Supplement, S118–S130, the Proceedings of the Twelfth Annual DFRWS Conference; 2012.

Fairbanks KD, Lee CP, Owen III HL. Forensic implications of ext4. In: proceedings of the sixth annual workshop on cyber security and information intelligence research. ACM; 2010. p. 22.

Garfinkel SL. Automating disk forensic processing with SleuthKit, XML and Python. In: IEEE systematic approaches to digital forensics engineering; May 2009. p. 73–84.

Garfinkel S, Nelson AJ, Young J. In: A general strategy for differential forensic analysis. Digital Investigation 9, Suppl (0), S50–S59, Proc Twelfth Annu DFRWS Conf; 2012.

Haass JC, Ahn G-J, Grimmelmann F. ACTRA: A case study for threat information sharing. In: Proceedings of the 2nd ACM Workshop on Information Sharing and Collaborative Security. WISCS '15. New York, NY, USA: ACM; 2015. p. 23–6.

Halcrow M. eCryptfs: a stacked cryptographic filesystem. Linux J Apr 2007;0(156):54–8. URL, http://www.linuxjournal.com/article/9400.

Halderman JA, Schoen SD, Heninger N, Clarkson W, Paul W, Calandrino JA, et al. Lest we remember: cold-boot attacks on encryption keys. Commun ACM May 2009;52(5):91–8.

Kim D, Park J, Lee K-g, Lee S. Forensic analysis of Android phone using ext4 file system journal log. In: future information technology, application, and service. Springer; 2012. p. 435–46.

Mell P, Grance T. The NIST definition of cloud computing: recommendations of the National Institute of Standards and Technology. NIST Special Publication; 2011. 800–145, http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

Nelson A. In: Advances in Digital Forensics VIII: 8th IFIP WG 11.9 International Conference on Digital Forensics, Pretoria, S. Afr, January 3–5, 2012; 2012. p. 51–65. Revised selected papers. Springer Berlin Heidelberg, Berlin, Heidelberg, Ch. XML Conversion of the Windows Registry for Forensic Processing and Distribution.

NPD Group. U.S. commercial channel computing device sales set to end 2013 with double-digit growth, according to NPD. Dec 2013. URL, https://www.npd.com/wps/portal/npd/us/news/press-releases/u-s-commercial-channel-computing-device-sales-set-to-end-2013-with-double-digit-growth-according-to-npd/.

NPD Group. Chromebooks are a bright spot in a stagnant B2B PC and tablet market, according to NPD. Aug 2015. URL, https://www.npd.com/wps/portal/npd/us/news/press-releases/2015/chromebooks-are-a-bright-spot-in-a-stagnant-b2b-pc-and-tablet-market-according-to-npd/.

NPD Group. NPD: U.S. B2B channel volumes rise to more than $62 billion, indicate steady and consistent growth for market. Mar 2015. URL, https://www.npd.com/wps/portal/npd/us/news/press-releases/2015/npd-us-b2b-channel-volumes-rise-to-more-than-62-billion-indicate-steady-and-consistent-growth-for-market/.

Olivier MS. On metadata context in database forensics. Digit Investig 2009;5(3):115–23.

Singh A, Lavine M, Turnbull B, Shiralkar T. Acer Aspire One netbooks: a forensic challenge. In: computer software and applications conference, 2009. COMPSAC '09. 33rd Annu IEEE Int, 2; July 2009. p. 404–9.

Swartz J. Apple loses more ground to Google's Chromebook in education market. Jan 2016. USA Today online URL, http://www.usatoday.com/story/tech/news/2016/01/11/apple-loses-more-ground-googles-chromebook-education-market/78323158/.

Thorpe S, Ray I, Grandison T, Barbir A. Cloud log forensics metadata analysis. In: Computer software and applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual. IEEE; 2012. p. 194–9.

United States Army, Chief Information Officer/G-6. Thin/zero client computing reference architecture version 1.0. Mar 2013. http://ciog6.army.mil/Portals/1/Architecture/ApprovedThinClient-ZeroComputingReferenceArchitecturev1-0_14Mar13.pdf.