Forensic Collection and Analysis of Thumbnails in Android*

Ming Di Leom[1] **Christian Javier D'orazio**[1] **Gaye Deegan**[2] **Kim-Kwang Raymond Choo**[1]
leomy009, dorcj002@mymail.unisa.edu.au Gaye.Deegan, Raymond.Choo@unisa.edu.au

[1]Information Assurance Research Group, University of South Australia, GPO Box 2471, Adelaide, SA 5001, Australia.
[2]School of IT & Mathematical Sciences, University of South Australia, GPO Box 2471, Adelaide, SA 5001, Australia.

Abstract

JPEG thumbnail images are of interest in forensic investigations as images from the thumbnail cache could be intact even when the original pictures have been deleted. In addition, a deleted thumbnail is less likely to be fragmented due to its small size. The focus of existing literature is generally on the desktop environment. Considering the increasing capability of smart mobile devices, particularly Android devices, to take pictures and videos on the go, it is important to understand how thumbnails can be collected from these devices. In this paper, we examine and describe the various thumbnail sources in Android devices and propose a methodology for thumbnail collection and analysis from Android devices. We also demonstrate the utility of our proposed methodology using a case study (e.g. thumbnails could be recovered even when the file system is heavily fragmented). Our findings also indicate that collective information obtained from the recovered fragmented JPEG image (e.g. metadata) and the thumbnail could be akin to recovering the full image for forensic purposes.

Keywords

Android forensic, forensic recovery, mobile forensics, thumbcache, thumbnail recovery

1. Introduction

Thumbnail is smaller representation of a larger media file such as picture and video, and has been used as evidence in a number of court cases in jurisdictions such as Australia, United Kingdom and United States. Quick, Tassone and Choo (2014, pp.1-2) also noted that '[i]n many cases, it is the thumbnail image alone that has been the evidence presented to court'. As the resolution of digital cameras increases, picture size and storage requirement also increases. To reduce the storage requirement and increase efficiency, operating system generally renders the thumbnail cache when a user is browsing the computer. Similarly,

* Copyright 2015 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE. This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder. Published version is available at https://doi.org/10.1109/Trustcom.2015.483

thumbnail cache can also facilitate forensic and digital investigations as the investigators can view thumbnail images significantly faster than the original images.

In this paper, we describe a thumbnail forensic recovery process for Android devices. We then demonstrate the utility of our process even in the event that the file system is no longer accessible and that we could link the recovered thumbnail and the associated fragmented deleted picture (in JPEG format) taken using the Android device's camera. In other words, the investigator would have the picture (albeit in lower quality) and the associated metadata (e.g. identifying previous whereabouts or accomplice of a terrorist suspect and determine whether a child pornography suspect possess illegal content).

1.1 Related work

In a recent work, Quick, Tassone and Choo (2014) provide a detailed overview of thumbnail stores for Windows platform (from Windows 95 to Windows 8) as well as the tools that can be used to view the thumbnails. The researchers also proposed an operational methodology for thumbnail analysis and a reporting and visualisation methodology and software prototype to present the findings from the thumbnail analysis. Other related work includes Matt (2012) which demonstrates how to recover thumbnail cache from Windows Vista machines and Hurlbut (2005) on machines running Windows Me to Windows XP using FTK (AccessData).

Other researchers (Parsonage 2012; Morris, SLA 2013) have also investigated thumbnail cache behaviour in Windows Vista and 7, and Windows 7 and Ubuntu Linux machines respectively. They demonstrate how the thumbnail is generated under user interaction with the OS. In one of few work for mobile devices, Hoog (2011) presented their preliminary study of thumbnail cache folder in Android devices, which required the file system to be accessible. Hoog's study did not include thumbnail cache behaviour, specifically how user action affects the creation of thumbnail cache. Existing thumbnail publications are summarised in Table 1.

The need to have a detailed understanding of thumbnail cache structure in order to facilitate automated extraction is highlighted by Hoog (2011). Existing literature generally focus on desktop operating system (OS). Considering the increasing ubiquity of mobile devices and that they are becoming a popular alternative to desktop, this paper aims to contribute to a better forensic understanding of thumbnail cache on Android devices.

Publications	**Platform**
Hurlbut (2005)	Windows Me to Windows XP
Hoog (2011)	Android
Matt (2012)	Windows Vista
Parsonage (2012)	Windows Vista and 7
Morris (2013)	Windows 7, Ubuntu Linux, and Kubuntu Linux
Quick et al. (2014)	Windows 95 to Windows 8

Table 1: Thumbnail publications by platform.

1.2 Contribution and outline of paper

In this paper, we propose a methodology for thumbnail collection and analysis from Android devices. We then demonstrate the utility of the methodology using a case study. In our case study, we first determine the characteristics of thumbnail in order to customise existing file carving tools to recover thumbnail from the forensic image in an efficient manner (e.g. by reduce the number of irrelevant files). Previous (Siciliano 2012; Honan 2013; The Guardian 2013; McColgan 2014; Schwamm 2014; Simon & Anderson 2015) have shown that performing factory reset on Android devices does not remove the actual content of data. Therefore, we demonstrate that it is possible to recover thumbnails even after the photos have been deleted, a factory reset has been undertaken by a user, or a corrupted file system.

Previous studies (Morris, S & Chivers 2011; Parsonage 2012; Quick, Tassone & Choo 2014) focusing on thumbnail cache behaviour in Microsoft Windows platform have shown thumbnail can be created in thumbcache without the original picture being viewed. This implies presence of thumbnail could not prove a user has knowledge of original picture in question. However, our experiment in Android platform (described in Section 4.3 How user actions affect creation of thumbnail cache?) shows that a certain size of thumbnail is only created after the picture has been viewed. This could possibly indicate that the user knew the existence of the picture in question.

The remainder of this paper is organised as follows. Section 2. presents an overview of Android forensics. Section 3. A methodology for thumbnail collection and analysis from Android devices outlines our proposed thumbnail forensic collection and analysis methodology for Android devices. We then demonstrate the utility of the process in Section 4. . In Section 5. Discussion, we discuss the potential limitations of using thumbnail in forensics and our process. The last 6. Conclusion and future work concludes this paper and outlines future research opportunities.

2. Background

Android mobile devices typically consist of several partitions. Partitioning scheme may differ between Android devices due to vendor customisation but there are generally six partitions and each partition stores data specific to a particular function (see Table 2). This information is potentially useful to a forensic investigator as it allows the forensic investigator to focus only on the relevant partition during the evidence identification process. The majority of the user's data is stored in the "user data" (*/data*) partition, internal SDcard, and cache partitions.

Name	**Mount point**	**Description**
Recovery	N/A	Recovery mode
Boot	N/A	Linux kernel
System	/system	Operating system files, system applications
Cache	/cache	Cache files
User data	/data	User installed application
Internal SDcard (Media)	/mnt/sdcard /storage/sdcardX /data/media/X	User-accessible storage to store media files.

Table 2: Partition layout of typical Android device. Adapted from Vidas, Zhang and Christin (2011)

The /data partition stores user personal information (e.g. Google account), user-installed applications, and updated versions of built-in applications. During a factory reset, both the data and cache partitions are formatted and, optionally, the internal SDcard (also known as the media partition). Generally, the internal SDcard is the largest partition and stores media files (e.g. songs, pictures, and videos), which are accessible from a desktop via a USB connection.

Prior to Android 3.0 (Honeycomb), /data and media were two separate partitions. The media partition generally uses FAT32, which can be mounted and accessed from a host computer through USB Mass Storage (UMS), just like a USB flash drive. The problem with this layout is that the /data partition has a limited size since the majority of the storage space is allocated to the media partition. This limits the amount and size of applications a user can install. In Android Honeycomb, media becomes a subfolder in /data as /data/media, rather than a separate partition. A user needs to access the Android device from a host computer through the Media Transfer Protocol (MTP). Android 4.2 introduced multi- user support, where each user is assigned a subfolder in /data/media. The default user is assigned to /data/media/0.

Each new user is subsequently assigned to /data/media/10, /data/media/11, /data/media/12 and so on.

Data recovery can be undertaken using logical or physical acquisition techniques. The former allows the extraction of allocated data still accessible on the file system. The latter directly accesses the raw data in the storage medium without attempting to reconstruct the file system, as the file system usually deletes the file location (unallocated) without deleting the actual content, for efficiency (i.e. it is significantly faster to remove the link to the file location than the actual content).

3. A methodology for thumbnail collection and analysis from Android devices

We now present our proposed process of recovering thumbnail files from Android device (Figure 1) and map it to the digital forensic framework (McKemmish 1999).

Identify

In this step, we identify the potential evidence and the sources. In the case of thumbnail recovery, an investigator will need to identify the locations of the thumbnails and, if possible, the thumbnail size as knowing the size allows one to customise the file carving tool with better accuracy. We can determine the thumbnail resolution from previously recovered thumbnail generated by an Android device with similar camera specification (i.e. megapixel count). From the thumbnail resolution, we can estimate the average size of the thumbnail. Finding the average size allows us to customise the file carving tool. It is also necessary to check whether the device has been *rooted*, which will inform actions to be undertaken in the next step.

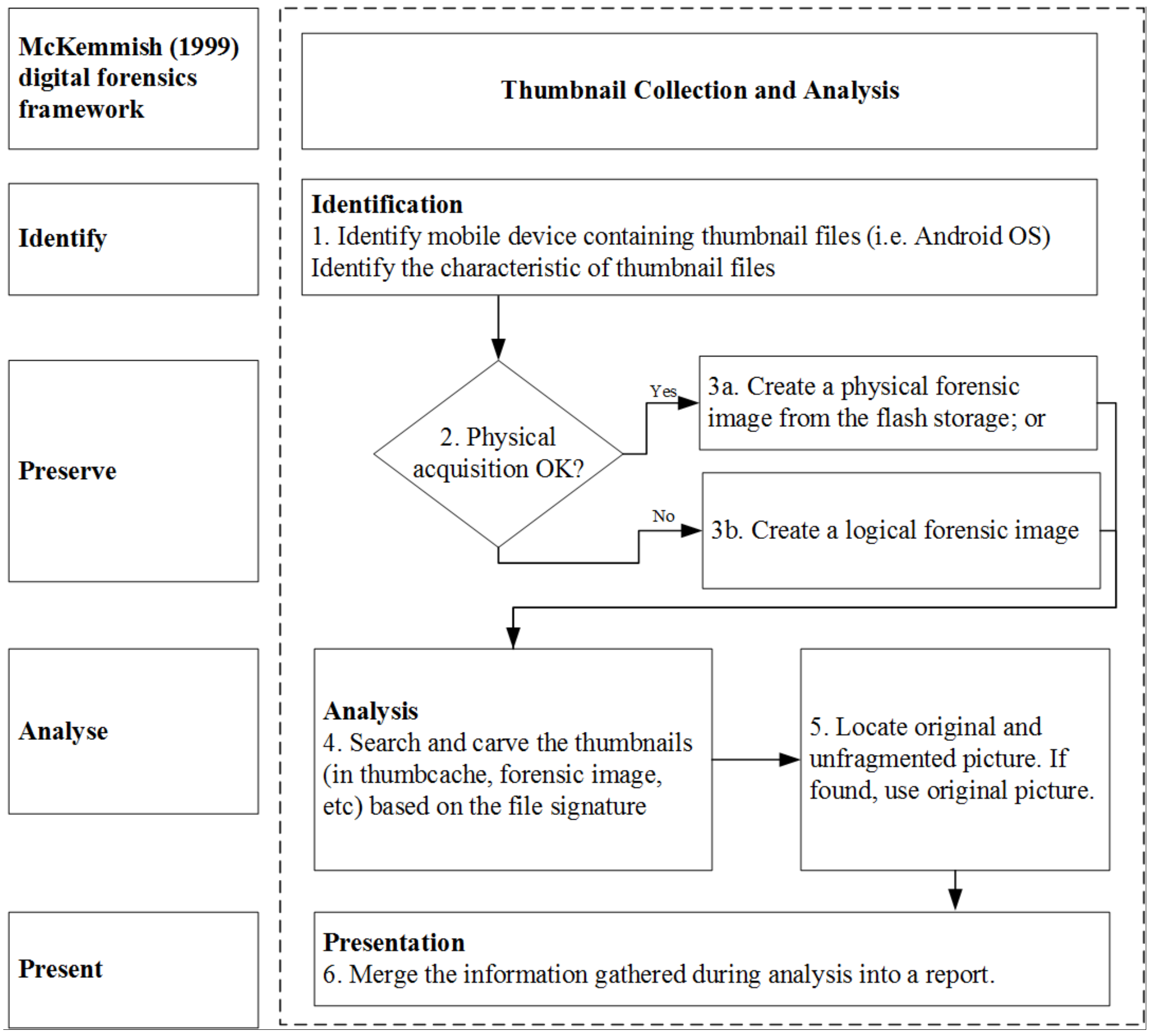

McKemmish (1999) digital forensics framework
Identify
Preserve
Analyse
Present
Thumbnail Collection and Analysis
Identification
1. Identify mobile device containing thumbnail files (i.e. Android OS) Identify the characteristic of thumbnail files
2. Physical acquisition OK?
Yes
3a. Create a physical forensic image from the flash storage; or
No
3b. Create a logical forensic image
Analysis
4. Search and carve the thumbnails (in thumbcache, forensic image, etc) based on the file signature
5. Locate original and unfragmented picture. If found, use original picture.
Presentation
6. Merge the information gathered during analysis into a report.

Figure 1: Thumbnail forensic recovery process for Android devices

Preserve

Wherever possible, a bit-for-bit copy of the flash memory should be undertaken. Otherwise, the forensic investigator could choose to acquire bit-for-bit copy of specific partition such as internal SDcard that are more likely to store thumbnail. The forensic image is hashed to ensure integrity throughout analysis. Bit-for-bit copy requires the Android device to have *root* access. Process of rooting often involves unlocking the bootloader and doing so will wipe the /data partition. However, in that scenario, the partition is not securely wiped and its previous content is recoverable through physical extraction (Wartickler 2012). Whether rooting is considered tampering with the evidence and consequently affect its admissibility is not part of discussion in this paper. Nonetheless, rooting may not be possible in certain model. In that case, thumbnail cache (thumbcache) files such as imgcache.0 could be logically extracted and preserved just like a cloned image.

Analyse

File carving tool can be used to locate thumbnail within the forensic image. The software should be customised according to the file signature of thumbnail file. This is necessary to reduce the number of irrelevant files recovered for easier analysis. This technique could also be employed to extract thumbnail from the recovered thumbcache file found in Android device. The recovered thumbnail can then be used to match existing fragmented file.

Presentation

Information gathered during analysis stage are documented and presented. This could comprise; (1) thumbnail picture which matches an existing file, (2) thumbnail picture which matches fragmented file, or (3) standalone thumbnail picture, in a report.

4. Case study

In this section, we conduct three case studies based on methodology. First case study focus on logical extraction of thumbnail. Second case study focus on physical extraction of thumbnail. Third case study focus on the behaviour of the thumbnail cache. Prior conducting these case studies, we performed factory reset beforehand. *USB debugging* is enabled in the device system settings. The Android mobile device is also *rooted* to access the raw data. USB debugging is required for forensic acquisition conducted on first and second case studies (Section 4.1 Logical extraction of thumbnail & 4.2 Physical extraction of thumbnail cache). Below table (Table 3) outlined the hardware and software specification.

Name/Model	**Description**	**Version/Specification**
Samsung Nexus S I9020T	Mobile device	Android 4.1.2 (rooted and USB debugging enabled)[1]
BusyBox (installed in mobile device)	Collection of Unix tools (e.g. nc, dd)	1.23.0, installed using BusyBox installer for Android v25 (Stericson 2014)
Gallery	Default Android media gallery app	1.1.40000
Dell Optiplex 960	Workstation	Intel Core 2 Quad Q9400 (2.66Ghz quad-core), 4GB RAM, 150GB hard disk, Windows 7 64-bit
Oxygen Forensic Suite 2014 (Oxygen Forensics)	Forensic tool	6.2.1.103 (Trial)
Netcat/nc (Pond 2004)	Forensic tool	1.11

[1] More than 2 years of usage. Previously equipped with custom ROM Android 4.4. Downgraded to stock Android 4.1.2 (Google 2014a).

HxD (Hörz 2009)	Hex editor	1.7.7.0
Scalpel (machn1k 2013)	File carver	2.0
ExifTool (Harvey 2014)	JPEG EXIF metadata viewer	9.69

Table 3: Hardware and software specification

Below (Table 4) shows the file system type of each partition in the test device:

Name	**File system**
Recovery	YAFFS2
Boot	
Cache	
System	EXT4
User data	
Internal SDcard/Media	FAT32

Table 4: Partition's file system

4.1 Logical extraction of thumbnail

In this case study, we capture 10 pictures using the mobile device as baseline pictures (Table V: File No. 21-30). Pictures from Gallery app are viewed and then deleted. After that, the partition /storage/sdcard0 where pictures are most likely to be stored is duplicated into our workstation using

```
$ adb pull sdcard/
```

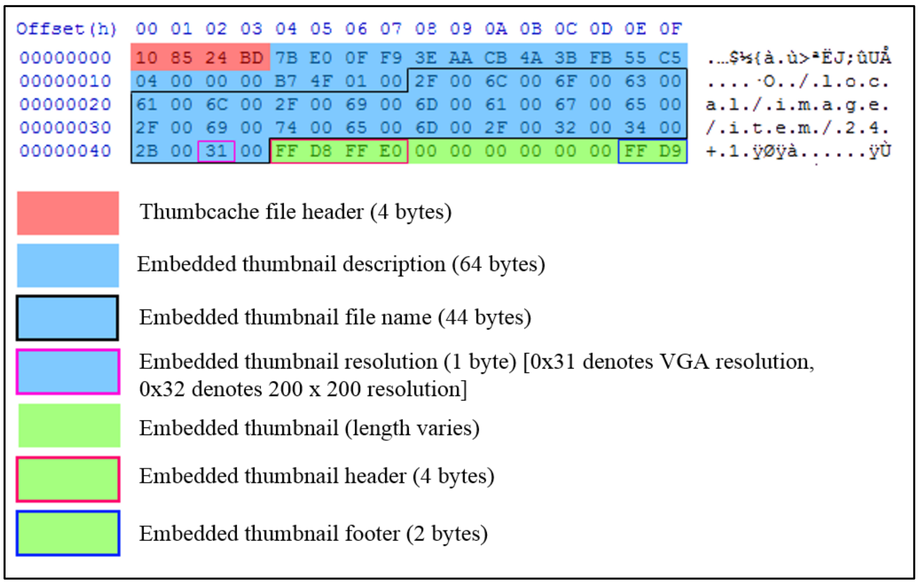
We identified the locations of the thumbnail cache (thumbcache) files, namely the thumbnail is embedded inside a JPEG file, and for thumbnails generated and used by Gallery app, the cache files can be found at /sdcard/Android/data/com.google.android.gallery3d.cache/img cache.0.

No.	**Filename**	**File size**	**Resolution**
21	IMG_20150117_175812.jpg	1,860	1920 x 2560
22	IMG_20150117_175852.jpg	1,593	1920 x 2560
23	IMG_20150117_175911.jpg	1,937	1920 x 2560
24	IMG_20150117_175930.jpg	1,281	2560 x 1920
25	IMG_20150117_175950.jpg	2,518	1920 x 2560
26	IMG_20150117_180001.jpg	2,524	1920 x 2560

27	IMG_20150117_180024.jpg	1,083	1920 x 2560
28	IMG_20150117_180050.jpg	1,393	1920 x 2560
29	IMG_20150117_180218.jpg	2,014	1920 x 2560
30	IMG_20150117_180249.jpg	1,379	1920 x 2560

Table 5: Baseline pictures

Thumbcache Viewer (Kutcher 2014a) and Thumbs Viewer (Kutcher 2014b) could not detect any thumbnail in File No. 31 since in both cases the software is designed to support thumbnail cache generated by Microsoft Windows OS, thus the tools are incompatible. We proceed to inspect File No. 31 using HxD, a hex editor. Figure below (Figure 2) shows file structure of sample thumbcache file (imgcache.0). The first 4 bytes (red-highlighted) contains header information as identifier for thumbcache file. The next 64 bytes (blue-highlighted) contains description of the following thumbnail. The description contains filename as identifier of the thumbnail inside the thumbcache. Note this is different from filename of the original picture. Next is the actual thumbnail data (green-highlighted). The size varies even with similar resolution. It includes header and footer. After that is description for the next thumbnail and its thumbnail data (as portrayed in Table 6 below). Thumbcache file does not have a unique footer value, rather the value is footer value of the last thumbnail.



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000  10 85 24 BD 7B E0 0F F9 3E AA CB 4A 3B FB 55 C5  ..…$½{à.ù>ªËJ;ûUÅ
00000010  04 00 00 00 B7 4F 01 00 2F 00 6C 00 6F 00 63 00  ....·O../.l.o.c.
00000020  61 00 6C 00 2F 00 69 00 6D 00 61 00 67 00 65 00  a.l./.i.m.a.g.e.
00000030  2F 00 69 00 74 00 65 00 6D 00 2F 00 32 00 34 00  /.i.t.e.m./.2.4.
00000040  2B 00 31 00 FF D8 FF E0 00 00 00 00 00 00 FF D9  +.1.ÿØÿà......ÿÙ
```

🟥	Thumbcache file header (4 bytes)
🟦	Embedded thumbnail description (64 bytes)
🟦	Embedded thumbnail file name (44 bytes)
🟦	Embedded thumbnail resolution (1 byte) [0x31 denotes VGA resolution, 0x32 denotes 200 x 200 resolution]
🟩	Embedded thumbnail (length varies)
🟩	Embedded thumbnail header (4 bytes)
🟩	Embedded thumbnail footer (2 bytes)

Figure 2: Internal structure of thumbcache file (imgcache.0).

Thumbcache header	Thumbnail 1 description	Thumbnail 1 data	Thumbnail 2 description	Thumbnail 2 data

Table 6: Internal structure of thumbcache file (imgcache.0) with multiple thumbnails.

We attempt to extract a thumbnail (File No. 32) by manually searching the header (**FF D8 FF E0**) and the footer (**FF D9**). We could recover the rest of the thumbnails manually using the hex editor, but file carving tool can help to automate this process. So, we extract the rest of the thumbnails using Scalpel, a file carving tool due to its ease of configuration and the customisation available fits our purpose.

Scalpel can be used through this command:

```
C:\>scalpel –v –c conf/scalpel.conf -o output_directory forensic_image
```

Where:

-v = verbose

-c = configuration file

-o = directory to store recovered files

The default configuration file is located at *conf* folder named *scalpel.conf*. The configuration file contains rules on file carving in each line. Default configuration had all the rules commented out so running Scalpel using that configuration will not recover anything. Each rule describes the file extension, whether the header and footer are case sensitive, the minimum and maximum file size, and the header and footer value.

In this case, we use following rule:

```
jpg     y       1000: 500000     \xff\xd8\xff\xe0   \xff\xd9
```

- ***jpg*** File extension.
- ***y*** Header and footer is case sensitive. Use '*n*' for case insensitive.
- ***1000: 500000*** Carve only file size between 1,000 bytes to 500,000 bytes. Ignore file outside of this range.
- ***\xff\xd8\xff\xe0*** Header value. \x is the representation for hexadecimal character. Use \? to match any byte value (wildcard).
- ***\xff\xd9*** Footer value. Optional.

Judging by the thumbnail resolution (VGA), we estimated the thumbnail size should be at least 1KB and less than 500KB. By using the above rule, we managed to extract 20 thumbnails. From the result (Table 7), we can observe that the Gallery app generate VGA (640 x 480) sized and 200 x 200 thumbnails for every picture.

We notice although File No. 21-30 are taken in portrait orientation (except File No. 24) (Table 5), all thumbnails are in landscape (Table 7 & Table 8).

A VGA resolution thumbnail that matches with its 200 x 200 resolution counterpart (Table 8) would have similar file name (**/local/image/item/00**+1) except for the last number (/local/image/item/00+**1**). Value "1" denotes the thumbnail is VGA resolution, while "2" denotes 200 x 200 resolution (described in more details in Figure 2).

No.	Image resolution	Filename	File size
31	N/A	imgcache.0	972.0
32	640 x 480	/local/image/item/24+1	83.9
33		/local/image/item/25+1	71.3
34		/local/image/item/26+1	83.5
35		/local/image/item/27+1	62.8
36		/local/image/item/28+1	129.2
37		/local/image/item/29+1	157.5
38		/local/image/item/30+1	39.8
39		/local/image/item/31+1	46.4
40		/local/image/item/32+1	114.8
41		/local/image/item/33+1	55.9
42	200 x 200	/local/image/item/33+2	7.7
43		/local/image/item/31+2	6.7
44		/local/image/item/32+2	16.8
45		/local/image/item/30+2	6.7
46		/local/image/item/29+2	20.8
47		/local/image/item/28+2	18.7
48		/local/image/item/27+2	11.4
49		/local/image/item/26+2	12.3
50		/local/image/item/25+2	11.8
51		/local/image/item/24+2	12.0

Table 7: Thumbnails extracted from imgcache.0 file.

Original (File No.)	Thumbnail (File No.)	Thumbnail (File No.)

21	32	51
22	33	50
23	34	49
24	35	48
25	36	47
26	37	46
27	38	45
28	39	43
29	40	44
30	41	42

Table 8: Original file and its thumbnail.

In order to extract the embedded thumbnail from a JPEG file, we use ExifTool. Although such tool is available, the manual method employed in extracting thumbnails from File No. 31 can work in this case as well.

The command used to extract thumbnail is as follows:

```
C:\> exiftool –b –ThumbnailImage input > output
```

*When using under Microsoft Windows, rename "*exiftool (-k).exe*" to "*exiftool.exe*".

Where:

-b = Output the requested data in binary format without tag names or descriptions.

-ThumbnailImage = Read thumbnail image

input = The location of original image.

output = The location to save the extracted thumbnail in .jpg extension.

Below (Table 9) shows the information on the extracted thumbnail stored in the 10 baseline pictures (File No. 21-30).

No.	**File size (KB)**	**Resolution**
52	13.1	320 x 240
53	13.4	
54	13.4	

55	13.0	
56	21.9	
57	27.6	
58	7.0	
59	6.5	
60	21.4	
61	7.8	

Table 9: Thumbnails extracted from original JPEG file.

4.2 Physical extraction of thumbnail cache

This section demonstrates method to physically extract the thumbnail from the raw forensic image. We first identify the /media[2] partition path (highlighted):

```
C:\> adb shell
shell@android:/ $ su
root@android:/ # ls -l /dev/block/platform/s3c-sdhci.0/by-name
lrwxrwxrwx root    root       2015-01-17 10:15 media -> /dev/block/mmcblk0p3
lrwxrwxrwx root    root       2015-01-17 10:15 system -> /dev/block/mmcblk0p1
lrwxrwxrwx root    root       2015-01-17 10:15 userdata -> /dev/block/mmcblk0p2
```

We then create forensic image of that partition using following commands:

```
C:\> adb forward tcp:5555 tcp:5555
C:\> adb shell
shell@android:/ $ su
root@android:/ # nc -l -p 5555 -e dd if=/dev/block/mmcblk0p3
```

(Note[3])

On another command prompt:

```
C:\> adb forward tcp:5555 tcp:5555
C:\> nc 127.0.0.1 5555 > mmcblk0p3.raw
```

We then use Scalpel to recover thumbnails from the forensic image. We customise the rule to target thumbnails only based on the results gathered in Section 4.1 Logical extraction of thumbnail (Table 7). We inspect the header information and determine the maximum file size

[2] The partition is also mounted as /storage/sdcard0, same partition used in Section 4.1 Logical extraction of thumbnail.

[3] *nc* and *dd* are installed through BusyBox.

of the thumbnails. The purpose is to determine the parameters that will be used in file carving. Our results show different type of thumbnail has different header value starting from 4th byte. Table 10 below illustrate the difference. Do note the header value for thumbnail cache shown in the table is not the header value of the thumbcache file, but rather the individual thumbnail stored inside the file.

Type	**Header**	**Footer**	**Maximum file size**
Thumbnail stored in thumbcache file	FF D8 FF E0	FF D9	157.5 KB
Embedded thumbnail in JPEG file	FF D8 FF DB		27.6 KB

Table 10: Header value and file size of thumbnail.

Based on the Table 10, we customise the rule to be as follows:

```
#1 rule
jpg y   1000:500000        \xff\xd8\xff\xe0             \xff\xd9
#2 rule
jpg y   1000:50000         \xff\xd8\xff\xdb             \xff\xd9
```

The first rule (*#1*) is to recover thumbnails from thumbnail cache file. 1KB is used as minimum size and 500KB is used as maximum size. The second rule (*#2*) is to recover embedded thumbnail. 1KB is used as minimum size and 50KB is used as maximum size.

Below (Table 11) shows the result of the thumbnails recovered.

Rule	**Thumbnail type**	**Thumbnails recovered**	**Percentage**
#1	200 x 200 resolution thumbnail in thumbcache	**10**/10	100%
	VGA resolution thumbnail in thumbcache	**3**/10 (**9**/10 if include fragmented thumbnail)	30%
#2	Embedded thumbnail in JPEG file	**10**/10	100%

Table 11: Recovery result.

The results show #2 rule is very effective at recovering thumbnails. The rule managed to recover thumbnail of all the test files (File No. 1 to 10). #1 rule also managed to recover all 200 x 200 resolution thumbnails of all the test files, but it is less successful on VGA resolution thumbnail. However, we still managed 90% if fragmented thumbnail is included.

This shows larger thumbnail is more likely to be fragmented. Nevertheless, the overall result shows that thumbnail is significantly less likely to be fragmented compared to original image.

4.3 How user actions affect creation of thumbnail cache?

Previous study (Hoog 2011) did not investigate how user action would affect creation of thumbnail cache. This section identifies the behaviour of the thumbcache when user interacts with the OS. To establish the behaviour of the Android thumbnail cache it is necessary to perform a variety of experiments; the experiments establish the way the thumbnail is generated based upon user activity. Prior to experiments in this section, the Android device is factory reset to clear the thumbcache. After factory reset, the device is connected to Wi-Fi and signed in with Google account. No third-party application nor any update are installed throughout this section. After each experiment, thumbcache file (located at /sdcard/Android/data/com.google.android.gallery3d.cache/imgcache.0) is copied to our workstation for analysis.

Experiment		**Result**
1	Take 10 pictures using default Camera app.	Found 8 VGA-sized thumbnails.
2	Launch Gallery app.	Previous thumbnails plus a 200x200 thumbnail.
3	Open the “Camera” (/sdcard/DCIM/Camera) folder.	Previous thumbnails plus 9 200x200 thumbnail.
4	View first picture.	Previous thumbnails plus 2 VGA-sized thumbnails.
5	View first to fifth picture.	No difference.
6	View first to tenth picture.	No difference.
7	Delete 5 pictures in odd number.	No difference.
8	Delete the remaining 5 pictures.	No difference.
9	Take 5 pictures.	Previous thumbnails plus 4 VGA-sized thumbnails.
10	Copy 31 pictures into “Pictures” (/sdcard/Pictures) folder.	No difference.

Table 12: Thumbcache behaviour in Android.

The experiments result (Table 12) shows VGA-sized thumbnail is generated when the picture is snapped but not for all pictures. When the gallery app is launched, a smaller size (200x200) thumbnail is generated. This thumbnail functions as the camera folder “cover”. When the folder is opened, all pictures are shown in “album view”, and at the same time the remaining 200x200 thumbnails of the 10 pictures are generated. The 2 remaining VGA-sized thumbnail is only generated after viewing the first picture. The thumbnail is not deleted even though the original image has been removed. No thumbnail is generated when there is new pictures is saved, that are not taken by the camera.

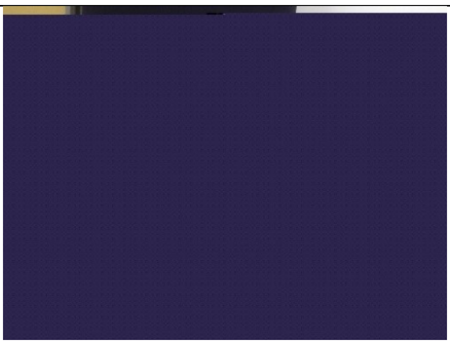
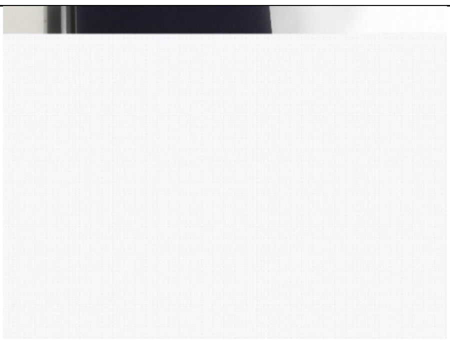
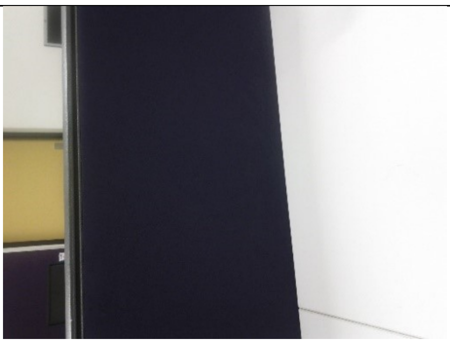
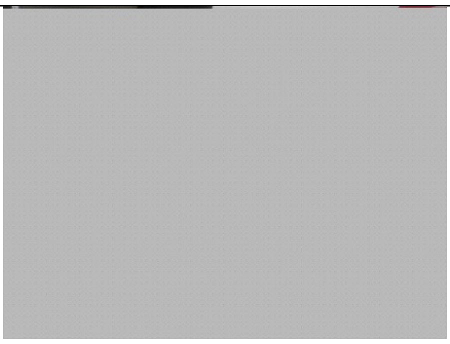
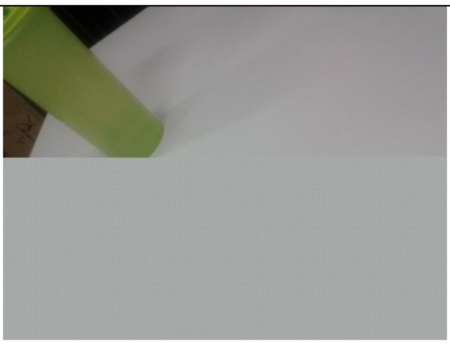
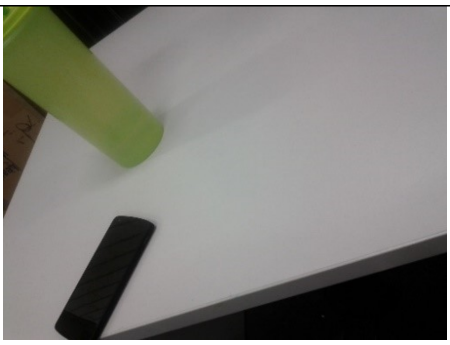
The implication of the results above is the possibility of using thumbnail as indication to determine whether the picture has been viewed or not.

5. Discussion

Recovered thumbnail can provide valuable visual clue to investigator. It can be matched with its original picture, especially the original picture’s metadata. The metadata is still intact even when the original picture is heavily fragmented, it still contains complete Exchangeable image file format (Exif) (Tachibanaya 1999) metadata. This is demonstrated in the following experiment. We take 10 pictures (File No. 1-10) using the mobile device and then perform physical acquisition of the mobile device using Oxygen Forensic Suite, a forensic tool. We attempt to recover them from the forensic image through following process:

1. Search for the first 8-byte value of the header in hexadecimal form of the intended picture. Mark the location the first byte of the found 8-byte header as START_OFFSET.
2. Select data block with the same value as the size of original picture (LENGTH=size of original picture).
3. Copy out the selected block of data.
4. Save the copied block of data with file extension “.jpg”.
5. Inspect the saved file using Windows Photo Viewer.
6. Verify MD5 hash of original and recovered picture.

We managed to recover File No. 1, 2, 7, 8, and 9 completely. The rest (File No. 3, 4, 5, 6, and 10) are fragmented (Table 13). In other words, out of 10 pictures, only half is not fragmented.

No.	Recovered	No.	Original
13		3	
14		4	
15		5	
16		6	

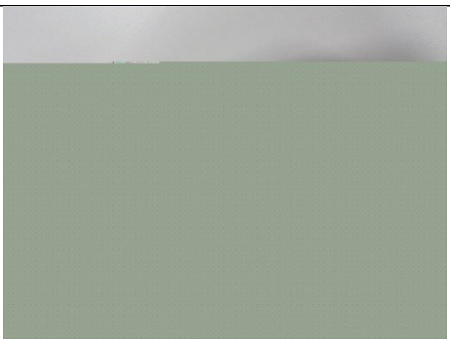
20		10	

Table 13: Recovered fragment and its original.

We then use ExifTool to determine existence of Exif metadata in File No 15. The results show that although heavily fragmented JPEG could only provide very limited visual clue but at least the EXIF metadata is not likely to be fragmented, thus recoverable. Below show excerpt of EXIF metadata in File No. 15 (the most fragmented among those shown in Table 13).

```
C:\>exiftool 15.jpg
ExifTool Version Number         : 9.69
File Name                       : 15.jpg
Directory                       : C:\Recovered fragment
File Size                       : 921 kB
File Type                       : JPEG
MIME Type                       : image/jpeg
Exif Byte Order                 : Little-endian (Intel, II)
Make                            : google
Camera Model Name               : Nexus S
Orientation                     : Rotate 90 CW
Software                        : JZO54K
Modify Date                     : 2014:06:19 18:06:06
Y Cb Cr Positioning             : Centered
Exposure Time                   : 1/33
F Number                        : 2.6
Exposure Program                : Aperture-priority AE
ISO                             : 50
Exif Version                    : 0220
Date/Time Original              : 2014:06:19 18:06:06
Create Date                     : 2014:06:19 18:06:06
```

By combining the metadata with a fully recovered thumbnail, these information are as valuable as original picture, since it still shows similar visual information, only with lower resolution. Example shown here, although a bank logo can be found in Figure 3, but it can refers to any office tower of the bank. In contrast, Figure 4 although is smaller, the location could be identified by the unique appearance of the shop on the left part. This can be further confirmed with location data from the EXIF metadata.

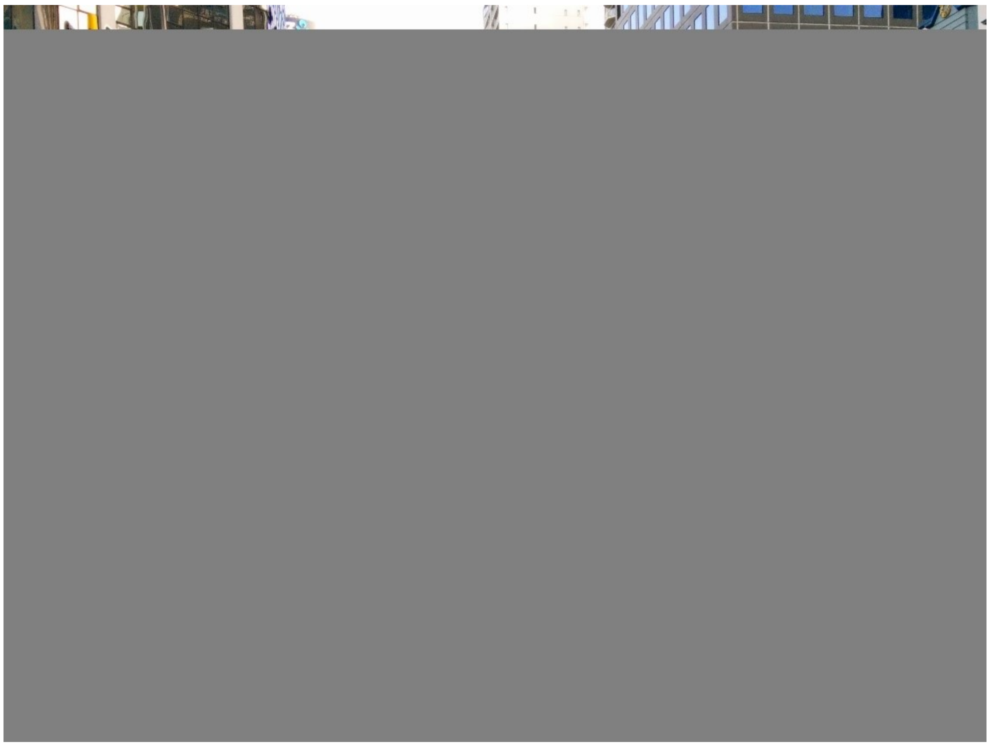


Figure 3: (Simulated) Heavily fragmented picture.

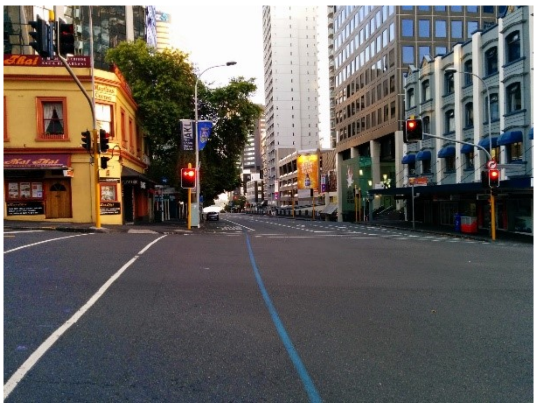


Figure 4: Non-fragmented. (Resized to VGA resolution to simulate a thumbnail)

Due to the small size of the thumbnail, it is less likely to be fragmented. In the event that we recover a fragmented thumbnail, it could still contain hints that could be used to identify the location of the photo taken.For instance, a simulated fragmented thumbnail depicted in Figure 5 contains popular landmarks (Royal Malaysia Police Headquarter and Merdeka Square) on the left part.

Using the size of the landmarks in the image, the distance from the landmark can be estimated. The shop building on the right part has a unique roof which would be recognized by individuals familiar with the locality. This could facilitate the investigation of, for example, a terrorist by helping to determine the whereabouts of a suspect.

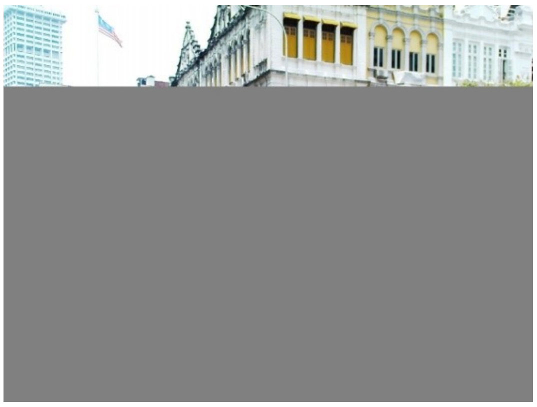


Figure 5: (Simulated) Fragmented thumbnail.

In cases such as child pornography investigations (Hillman, Hooper & Choo 2014), the thumbnail could be used to determine the existence of illegal content on a device even if the original images have been deleted. A thumbnail could also be used to determine the authenticity of the original photo, and whether the photo has been modified (Kee & Farid 2010). This is because image manipulation programs usually do not update the thumbnail after editing, leaving the original thumbnail still intact (Murdoch & Dornseif 2004).

6. Conclusion and future work

In this paper, we have described the location of thumbnail cache file "imgcache.0" and technique to extract thumbnails from that file. We also describe the file structure of the thumbcache file. We described the techniques to recover embedded thumbnail and the property of the embedded thumbnail. We demonstrated those techniques are effective in recovering thumbnail even when the file system is heavily fragmented. We also show the possibility of fragmented JPEG still holding important metadata and when link to thumbnail, is akin to recovering the full picture.

In this paper, we only demonstrated recovering thumbnail cache in Android generated by *Gallery* app. In future, we hope to extend our research to other Android gallery apps such as Photos app bundled with Google+ app (Google 2014b) and custom gallery apps shipped by vendors. We also hope to extend our research to newer Android versions. We also demonstrated the recovery method on a particular Android device. In theory, Android device with similar camera specification should have similar thumbnail size as well, while Android device equipped with camera that has higher resolution could result in larger thumbnail size. Thus, we also hope to extend our research to more Android devices.

Instead of relying on manual matching, linking the thumbnail to original image can be automated through computer algorithm to match the thumbnail to the fragmented JPEG. This idea is similar to Guo and Xu (2011) but that work focus on using thumbnail to rearrange JPEG fragments. There is also possibility of recovering fragmented thumbnail. However, we need to evaluate the feasibility of such approach in future work.

Although our experiments in this paper are conducted on Android mobile device, we believe that our proposed method could also apply to traditional desktop forensic. In Section 5. Discussion, we show that our method works well on bi-fragmented JPEG. Bi-fragmented is when a file is fragmented into two parts. Study (Garfinkel 2007) showed that it is the most common type of fragmentation in hard disk, thus showing the potential of our proposed method on desktop forensic.

References

AccessData *Forencis toolkit*, <http://accessdata.com/products/computer-forensics/ftk>.

Garfinkel, SL 2007, 'Carving contiguous and fragmented files with fast object validation', *Digital Investigation,* vol. 4, pp. 2-12.

Google 2014a, *Factory images for nexus devices*, viewed 11 August 2014, <https://developers.google.com/android/images>.

Google 2014b, *Google+ for Android*, Google Play Store, viewed 16 August 2014, <https://play.google.com/store/apps/details?id=com.google.android.apps.plus>.

Guo, H & Xu, M 2011, 'A Method for Recovering JPEG Files Based on Thumbnail', *International Conference on Control, Automation and Systems Engineering*, IEEE, pp. 1-4.

Harvey, P 2014, *ExifTool*, viewed 16 August 2014, <http://owl.phy.queensu.ca/~phil/exiftool/>.

Hillman, H, Hooper, C & Choo, K-KR 2014, 'Online child exploitation: Challenges and future research directions', *Computer Law & Security Review,* vol. 30, no. 6, pp. 687-698.

Honan, M 2013, *Break out a hammer: You'll never believe the data 'wiped' smartphones store*, Wired, viewed 19 April 2015, <http://www.wired.com/2013/04/smartphone-data-trail/>.

Hoog, A 2011, *Android Forensics: Investigation, Analysis and Mobile Security for Google Android*, Syngress.

Hörz, M 2009, *HxD - Freeware Hex Editor and Disk Editor*, viewed 13 August 2014, <http://mh-nexus.de/en/hxd/>.

Hurlbut, D 2005, *Thumbs DB files forensic issues*, AccessData, viewed 28 August 2014, <http://repo.zenk-security.com/Techniques%20d.attaques%20%20.%20%20Failles/THUMBS%20DB%20FILES%20FORENSIC%20ISSUES.pdf>.

Kutcher, E 2014a, *Thumbcache Viewer*, GitHub, viewed 16 August 2014, <https://github.com/thumbcacheviewer/thumbcacheviewer>.

Kutcher, E 2014b, *Thumbs Viewer*, GitHub, viewed 16 August 2014, <https://github.com/thumbsviewer/thumbsviewer>.

machn1k 2013, *Scalpel-2.0*, GitHub, viewed 15 August 2014, <https://github.com/machn1k/Scalpel-2.0>.

Matt 2012, *Analyzing Thumbcache*, viewed 28 August 2014, <http://escforensics.blogspot.com/2012/11/analyzing-thumbcache.html>.

McColgan, J 2014, *Tens of thousands of Americans sell themselves online every day*, AVAST Software, viewed 10 July 2014, <https://blog.avast.com/2014/07/08/tens-of-thousands-of-americans-sell-themselves-online-every-day/>.

McKemmish, R 1999, 'What is forensic computing?', *Trends and Issues in Crime and Criminal Justice,* vol. 118, pp. 1-6.

Morris, S & Chivers, H 2011, *An Analysis of the Structure and Behaviour of the Windows 7 Operating System Thumbnail Cache*, University of Strathclyde, Glasgow, UK.

Morris, SLA 2013, 'An Investigation into the identification, reconstruction, and evidential value of thumbnail cache file fragments in unallocated space', PhD thesis, Cranfield University,, Shrivenham, Oxfordshire, UK.

Oxygen Forensics *Oxygen Forensic® Suite - Mobile forensic software for cell phones, smartphones and other mobile devices*, viewed 10 August 2014, <http://web.archive.org/web/20150312163430/http://www.oxygen-forensic.com/en/products/oxygen-forensic-suite/features [archived]>.

Parsonage, H 2012, *Under My Thumbs - Revisiting Windows Thumbnail Databases and Some New Revelations About the Forensic Implications*, viewed 28 August 2014, <http://computerforensics.parsonage.co.uk/downloads/UnderMyThumbs.pdf>.

Pond, W 2004, *netcat (Windows)*, viewed 14 August 2014, <http://web.archive.org/web/20140725175856/http://www.securityfocus.com/tools/139 [archived]>.

Quick, D, Tassone, C & Choo, K-KR 2014, 'Forensic analysis of Windows thumbcache files', *Americas Conference on Information Systems*, Association for Information Systems.

Schwamm, R 2014, 'Effectiveness of the factory reset on a mobile device', Naval Postgraduate School, Monterey, California.

Siciliano, R 2012, *I Found Your Data on That Used Device You Sold*, McAfee, viewed 19 April 2015, <http://web.archive.org/web/20130623083846/http://blogs.mcafee.com/consumer/i-found-your-data-on-that-used-device-you-sold [archived]>.

Simon, L & Anderson, R 2015, 'Security Analysis of Android Factory Resets', *Mobile Security Technologies Workshop*, IEEE, pp. 1-10.

Stericson, S 2014, *BusyBox*, Google Play Store, viewed 27 August 2014, <https://play.google.com/store/apps/details?id=stericson.busybox>.

Tachibanaya, TZ 1999, *Description of Exif file format*, Personal Information Architecture, MIT Media Laboratory, viewed 17 August 2014, <http://www.media.mit.edu/pia/Research/deepview/exif.html>.

The Guardian 2013, *Recycled mobile phones retain previous owner data*, viewed 19 April 2015, <http://web.archive.org/web/20140529182505/http://www.theguardian.com/media-network/partner-zone-infosecurity/mobile-phones-previous-owner-data [archived]>.

Vidas, T, Zhang, C & Christin, N 2011, 'Toward a general collection methodology for Android devices', *Digital Investigation,* vol. 8, pp. S14-S24.

Wartickler 2012, *[GUIDE] Internal Memory Data Recovery - Yes We Can!*, XDA Developers, viewed 3 December 2014, <http://www.xda-developers.com/android/restore-galaxy-nexus-internal-memory-after-bootloader-unlock-wipe/>.