Lessons Learned Writing Computer Forensics Tools and Managing a Large Digital Evidence Corpus

By

Simson Garfinkel

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2012 USA

Washington, DC (Aug 6th - 8th)

Contents lists available at SciVerse ScienceDirect

Digital Investigation

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

Lessons learned writing digital forensics tools and managing a 30TB digital evidence corpus

Simson Garfinkel*

Naval Postgraduate School, Computer Science, 900 N Glebe Rd, Arlington, VA 2203, United States

Keywords:
Digital forensics
Lessons learned
Digital corpora

ABSTRACT

Writing digital forensics (DF) tools is difficult because of the diversity of data types that needs to be processed, the need for high performance, the skill set of most users, and the requirement that the software run without crashing. Developing this software is dramatically easier when one possesses a few hundred disks of other people's data for testing purposes. This paper presents some of the lessons learned by the author over the past 14 years developing DF tools and maintaining several research corpora that currently total roughly 30TB.

Published by Elsevier Ltd.

1. Introduction

As the field of digital forensics (DF) continues to grow, many people find themselves engaged in the once obscure practice of writing DF software. Few of today's forensic tool developers have formal training in software development or design—many do not even see themselves as programmers. They say that they are writing "scripts," not programs, apologize that their efforts are neither efficient nor elegant, and explain that their scripts are simply stopgap measures until professional, seasoned programmers can be hired to sort things out.

This situation is reminiscent of the Internet's early days. The people we now call Internet "pioneers" did not regard themselves so grandly. The tradition of titling Internet specifications as "Request For Comment" started because "[m]ost of us were graduate students and we expected that a professional crew would show up eventually to take over the problems we were dealing with" (Reynolds and Postel, 1987). Instead, the graduate students became the professionals.

Likewise, no professional crew is coming to take over our DF problems. If we want better tools, we must build them ourselves.

By *digital forensics software* I mean software that is used to analyze disk images, memory dumps, network packet captures, program executables, office documents, web pages and container files. That is, it is software that can be used to analyze any information that might be found on a computer or sent over a network during the course of an investigation. Fortunately, the same tools can frequently be used with many different modalities—the EXIF from a JPEG can be usefully analyzed no matter whether the JPEG was extracted from disk sectors or reassembled from IP packets.

I also take the word *investigation* to be quite broad: although some practitioners focus on the use of DF tools in criminal investigations, these tools are also used in civil investigations, internal investigations, and even audits, all of which have different standards for chain-of-custody, admissibility, and scientific validity. This broad scope means that many DF tools and techniques that might be improper to use in one context might be routine in others.

Despite the expansiveness of this definition, some areas are clearly outside DF. I exclude multimedia forensics—for example, the analysis of a JPEG to determine if it has been altered—agreeing with Böhme et al. (2009) that multimedia forensics is not digital forensics. This article also ignores advances in search and text retrieval and natural language processing, as those are simply not my area of research. Rather than being a jack of all trades, I attempt to

* Tel.: +1 617 876 6111.
E-mail address: simsong@acm.org.

1742-2876/$ – see front matter Published by Elsevier Ltd.
doi:10.1016/j.diin.2012.05.002

develop software that focuses on structure and metadata, rather than content, and has a plug-in architecture, so that expert developers in those fields can transparently meld their software with mine.

This article is written in the first person and is necessarily personal. I hope it will allow others to learn not merely from my research papers, but from my unreported experiences as well as my mistakes that were made along the way.

1.1. Outline

This concludes the introduction. Section 2 explains why DF tools are harder to develop than other kinds of software; it also argues why market conditions justify such software development directly funded by governments. Section 3 presents problems that I encountered in creating the Real Data Corpus. Section 5 lists some other articles that you should also read. Section 6 concludes.

2. Digital forensics is different

Several aspects of DF software development make it fundamentally different than other areas. These include: data diversity; data scale; temporal diversity; human capital; and the so-called "CSI effect."

2.1. The challenge of data diversity

A fundamental distinction between DF and other kinds of software is the range of data that must be analyzed. Most software development is confined to a particular problem domain. DF is concerned with the totality of information that can be stored or transmitted using computer systems: any computer used for any purpose might need to be analyzed as part of a criminal investigation, civil lawsuit, or military operation. As there are many more developers writing general-use software than DF tools, the percentage of data types that can be analyzed, by DF tools is likely to decrease over time.

The need to process incomplete or corrupt data further complicates the task of the DF developer. In most computing contexts tools validate their inputs and refuse to run if the data are inconsistent. DF tools cannot refuse to run—they must make a best effort attempt to show whatever data are present on the subject media.

DF tools also need to determine *why* data will not validate. Sometimes it is because data have been corrupted by partial overwriting. Other times there have been deliberate attempts at falsification or information hiding. Minor inconsistencies can indicate tampering. Unfortunately much of the redundant information on modern computers is not validated for internal consistency by modern DF tools. For example there are multiple time stamps associated with activities on a Windows systems, but most programs concentrate on just a few time stamps, ignoring others (Nelson, 2012). Such inconsistencies should be detected and reported.

Rather than directly detecting inconsistent information, an alternative is to eliminate data that are consistent or otherwise explainable. That is, eliminate the truth and the improbable, and whatever remains must be impossible—and therefore falsified. Data that are improbable should be examined for steganography.

2.2. Data scale

A second problem in DF tool development is the amount of data that must be processed and the never-ending battle with storage and performance bottlenecks. These problems result from the fact that investigators invariably need to analyze recently purchased computer systems. We are thus using top-of-the-line systems to analyze top-of-the-line systems, and we typically need to analyze in hours (or days) what a subject spent weeks, months, or even years assembling. We will never get ahead of the performance curve. When we finally move to cloud-based analysis, we will need to analyze the multi-terabyte results of investigating cloud-based crime.

Somewhat surprisingly, it has been difficult to apply many "big data" solutions from other fields to DF. For example, most cluster computers are designed for problems such as weather modeling and finite element analysis, which require large working sets in memory and high-speed communications between nodes, but which do not ingest terabytes of heterogeneous data for each run. High-energy physics experiments generate large amounts of data, but the data can be locally processed and reduced. Also, state and local investigators do not have the budgets or personnel that would let them adapt the data management practices of the world's leading physics laboratories.

One solution to the performance bottleneck is to adopt *sub-linear algorithms* that operate by sampling data. Sampling is a powerful technique and can frequently find select data on a large piece of media with a high degree of precision: at NPS we developed a technique for determining the amount of encrypted data on a 2 Terabyte (TB) hard drive in less than 10 min (Garfinkel et al., 2010). But sampling cannot prove the *absence* of data: the only way to establish that there are no written sectors on a hard drive is to read every sector.

I believe that scale and performance bottlenecks dictate that forensic researchers develop new approaches that rely on the inherent advantages of governments, such as the ability to correlate email addresses—just as Automated Fingerprint Identification Systems (AFIS) revolutionized law enforcement in the 1990s (Snow 2007, p.111). Such capabilities can reduce the need to perform traditional forensic processes.

2.3. Temporal diversity: the never-ending upgrade cycle

Many computer users have learned that upgrades are a disruptive process that need to be carefully managed. As a result, many organizations run out-of-date operating systems and only move to newer ones when they buy new hardware.

DF practitioners do not have this luxury. They must continually update their software, for the simple reason that examinations routinely involve the newest software and hardware on the market. In no other field of computing is it necessary to upgrade software the very moment that new software comes out—many websites work fine with browsers that are three years old, and vice versa. But when it comes to forensics, failure to upgrade invariably means failure to analyze.

When upgrading tools, there are two version numbers that need to be considered. The first is the tool's *target version*, the version of the operating system that it can analyze. The second is the tool's *host version*, the version of the operating system required to run the tool.

My practice is to upgrade my own computers as soon as new software comes out, for the simple reason that I would rather discover compatibility problems with my software than place the burden on my users. Upgrading my host system also makes it easier for me to support the host as a target.

To be clear, upgrading a DF tool to support a new host operating system does not mean dropping support for an older host operating system—many users cannot upgrade. Each operating system release therefore means supporting a broader set of platforms. And indeed, forensic software running on older host operating systems must support the most up-to-date targets.

Support of target operating systems is even more complicated. Not only must tools support new targets within days of their release, but they must continue to support old target releases *forever*. EnCase Forensic cannot run on Windows 3.1 but it must be able to analyze Windows 3.1 systems, since an investigator might come across an old computer system that needs analysis. The problem is worse for programs that analyze web services. Facebook and Google constantly change their HTML, JavaScript, JSON and XML formats. As a result, a computer that is seized today may have data from a hundred different versions of Facebook in its cache; all must be analyzed by a single tool.

We have two options. Either DF tools must become geometrically more complex over time, or they must adopt a combination of pattern matching, machine learning, and automated code analysis to infer what captured data *probably means*. The certainty (or lack of it) must then be shared with the examiner. An example of this approach can be found in Walls et al. (2011a).

2.4. Human capital demands and limitations

Hibshi et al. (2011) found that users of DF software come overwhelmingly from law enforcement, with little or no background in computer science. They are generally deadline-driven and over-worked. Examiners that have substantial knowledge in one area (e.g., NTFS semantics) will routinely encounter problems requiring knowledge of other areas (e.g., Macintosh malware) for which they have no training. Certifications and masters' degrees are helpful, but cannot fundamentally address the diversity problem as any examiner might reasonably be expected to analyze any information that their organization might possibly encounter on digital media. This is one of the reasons that it typically takes two years to train a DF examiner to the point of being able to work a case on their own, even after they have achieved some kind of certification.

Among developers, the human capital problem plays out in a different way. Data diversity means that developers require knowledge of all levels of the computing stack, from individual opcodes, multi-threading, the organization of processes and operating system structures, networking, and supercomputing. It is exceedingly hard to find individuals with such background—let alone pay what they are worth. As a result, many organizations that develop software need to train their own developers to the task. Coincidentally, it also seems to take about two years for a programmer to become proficient at developing computer forensics tools; I am not sure why this is the case.

2.5. The CSI effect

Much has been written regarding the so-called *CSI Effect*, a hypothesis that television shows portraying Crime Scene Investigations and forensic science cause juries, judges and even prosecutors to have unreasonable expectations regarding what forensic examinations can actually conclude (Shelton, 2008; Lawson, 2009). In recent years DF has become a staple of many such shows. On the screen nearly every DF investigator is trained on every tool; correlation is easy and instantaneous; there are never false positives; overwritten data can frequently be recovered; encryption can frequently be cracked; it is all but impossible to delete anything; and tools never crash.

Reality is not so kind. Overwritten data cannot be recovered (Wright et al., 2008) and modern encryption algorithms can only be decrypted with password cracking. As a result, most examiners spend their time looking for data that the suspect did not observe or neglected to erase. Such data rarely answers specific questions or establishes guilt.

Complicating the public's perception of DF is that fact that forensic tools perform functions that are fundamentally similar to programs like Windows Explorer, Outlook, and Google. That is, tools can list files, display email messages, and perform search. A person who is skilled in *using* computers may believe that they have a good understanding of forensic investigation and underestimate the complexity and uncertainty inherent in the process.

The head of an academic research lab once told me that recovering data from hard drives wasn't all that difficult, and if I really wanted to apply my talents, I should develop covert ways of downloading files from terrorist websites. The researcher was wrong. Terrorist web servers are *designed* to provide information to terrorists: the only difficulty is pretending to be one. Recovering data from hard drives typically involves decoding data that is fragmented or partially overwritten, and that can no longer be processed by the terrorists' own tools. That is fundamentally hard.

The differences between Windows Explorer and EnCase Forensic are not obvious to the uninitiated. DF is a difficult process that looks easy. This is not a good formula for continued funding. It is vital that we convey to those outside our field the scope of the technical difficulties that we face; if we do not, we are sure to see a decrease in both funding and future recruits.

2.6. The cost of development and the role of government

DF tool development is exceedingly expensive and the resulting software has a limited user base. The more sophisticated the analysis, the smaller the market. There is an obvious commercial market for file recovery tools, for example. But there is no market for tools that extract email addresses from fragments of Windows hibernation files and correlate them with email addresses from the address books of cell phones. Not surprisingly, few DF companies have been commercially successful. Some have an initial success with their product and scale up staff accordingly, only to discover that the initial release saturated the market. It is not that DF is an immature market with customers being mostly in the government: it is that DF is a mature market with development costs that are high and increasing. It is important to note that Guidance Software, despite a successful initial public offering in 2006, did not have a profitable quarter until the 3 months ending 2011-SEP-30, when it made just $510,000 on $27.26M in revenue—a profit margin of less than 2% (Guidance Software, 2011).

High development costs combined with a user base that is almost entirely within federal, state and local governments limits the applicability of the traditional commercial software development model. It may be more appropriate for development costs to be paid up front by governments, either with internally developed software or through contract vehicles, and then to freely distribute the resulting software to qualified users. This is why, after watching numerous DF companies (including my own) fail, I devoted my efforts to writing open source DF tools.

3. Lessons learned managing a research corpus

A key part of my research program has been the acquisition, curation, use, and distribution of DF data. This project started in 1998 and has expanded to include data from hard drives, cell phones, digital cameras and other devices. Today the corpus includes nearly a million redistributable files downloaded from US Government web servers, disk images from thousands of hard drives purchased around the world, and several terabytes of "realistic" scenarios manufactured by students. More details of the corpus can be found in Garfinkel et al. (2009).

This section describes the rationale for using real data in computer forensics research (§3.1), then presents some of the technical (§3.2) and policy (§3.3) issues that I encountered.

3.1. History: how I started acquiring used data

In 1998 I purchased six very obsolete used computers at a local computer store to test a multi-line telephone scanner Sandstorm Enterprises was developing. Before reformatting the hard drives, I discovered that the machines had sensitive information from previous users. I reported my finding to the store owner, who asked that I wipe the machines. I wondered whether or not other used equipment might contain sensitive data, and started purchasing used hard drives on eBay, at computer stores, and flea markets. I called this the "Drives Project."

I had amassed a personal collection of more than 150 hard drives when I entered the MIT Computer Science PhD program in the fall of 2002. Needing a research project, I decided to scavenge the drives for sensitive information left behind by previous users. I tried programs created for law enforcement such as EnCase and FTK and found that while they could recover deleted files and perform keyword searches, they were less useful for finding loosely defined "sensitive stuff." Shelat and I developed software that scanned for email addresses and credit card numbers, allowing us to select from the many drives, the few that contained the most spectacular collections of sensitive data. These were manually investigated and reported in Garfinkel and Shelat (2003). In Garfinkel (2005) I showed how usability failures in modern operating systems led directly to the data leakage.

My strategy of scanning for "sensitive stuff" had far-reaching consequences. Traditionally DF was used as a *conviction support tool*. That is, law enforcement seized computers belonging to people that were already suspects, and the computers, typically searched with a warrant, provided evidence of guilt. My new strategy re-purposed DF as a tool for *investigations and intelligence*, with the goal of finding leads by focusing on unusual signals that were likely to be probative. To use the language of machine learning, the detectors were set to a point where recall was low but accuracy was high. The implicit assumption was that significant data, once found, could be used to find additional significant data through correlation.

It turns out that I entered my PhD program at a pivotal time. A year following the terrorist attacks of September 11, 2001, there was a growing realization that the attacks had happened because a " 'wall' had been built between intelligence and law enforcement" organizations within the Federal Government (Shelby, 2002). The "wall" prevented "connecting the dots… in ways that good intelligence analysts are expected to do."

One reason that it's hard to connect the dots is that there are too many of them. My approach of searching hard drives for information that was sensitive and unusual created a system that found just a few dots, but the dots were easy to connect.

3.2. Corpus management—technical issues

I quickly learned that it is invariably easier to collect data than to analyze it. This section discusses some of the more interesting technical problems that arose and were solved between 2006 and 2012, more-or-less in the order that they were encountered.

3.2.1. Imaging ATA drives

Most of the drives that I personally purchased between 1998 and 2006 were ATA drives. Lacking a write-blocker and not needing chain-of-custody, I attached the drives to a computer through the on-board ATA interface and imaged the drives with the Unix *dd* command. Contrary to popular belief, I learned that ATA hardware supports hot-swapping of ATA drives while the system is running, making it easy to build a system that could image multiple drives at once, allowing me to swap drives without rebooting. This saved a substantial amount of time. *Lesson: read the documentation for the computer that you are using.*

Initially I kept disk images in raw format. When I ran out of storage, I decided to compress the images with *gzip*. I then wrote software that analyzed each compressed image as a single stream, using *zcat* to decompress the file and then processing the decompressed stream on standard input. This approach did not let me mount the file systems, but it was sufficient for finding sensitive information. This naturally lead me to the development of stream-based forensic processing that ignores files and instead relies on bulk data processing. *Lesson: make the most of the tools that you have and follow the technical innovations they force upon you.*

3.2.2. Automation as the key to corpus management

Once I had more than a few dozen drives it became clear that I would need to automate as much of the process as possible if I wanted to make any scientific use of the data that I was accumulating. I developed an accession process for each drive that involved automatically capturing the make, model, serial number, where the drive was purchased, for how much, the eBay auction ID, the ID of the seller, and so on, and automatically stored this information in a MySQL database. Over time I learned that information in a database is hard to move from system to system—it does not automatically follow the disk image. For this reason I started work developing an evidence file format in which I could embed arbitrary information. *Lesson: automation is key; any process that involves manual record keeping is going to introduce inaccuracies that will be hard to detect and correct. Lesson: useful data will outlive the system in which it is stored, so make provisions to move the data when you design the system.*

3.2.3. Evidence file formats

During the summer of 2005 I explored several approaches for storing arbitrary evidence and metadata in a single file. The obvious solution was Berkeley DB, but it did not have an appropriate open source license. GNU SDB did, but it created sparse files that had large “holes” in them and couldn’t be easily moved from one computer to another. The ZIP file format was probably the right solution, but ZIP32 was too limiting and I did not have a clean ZIP64 implementation. Instead of writing an open source ZIP64, which was the correct choice, I created my own container file format which I called AFF (Garfinkel et al., 2006).

Storing arbitrary name–value pairs was quite flexible, and it proved relatively easy to add chain-of-custody and strong cryptography to the format and the implementation (Garfinkel, 2009). In 2009 Michael Cohen contacted me and asked about addressing some of the performance bottlenecks caused by AFF’s simple design. I suggested that we abandon my container format and move to ZIP64, which we did (Cohen et al., 2009).

Since then I have largely abandoned AFF, as most of my users have standardized on the EnCase file format. Fortunately I am able to read and write the EnCase “E01” file format using libewf (Metz, 2011). *Lesson: avoid developing new file formats whenever possible. Lesson: kill your darlings.*

3.2.4. Crashes from bad drives

Occasionally my FreeBSD system would crash when imaging a faulty ATA drive. Analyzing the crashes, it appeared that parts of kernel memory had been overwritten. I concluded that the faulty drive was transferring data into the operating system using the ATA DMA facilities and that the OS was not defended against transfers to incorrect memory locations. This was an important discovery, because it likely meant that PC system memory could be imaged through specialized hardware attached to the ATA interface, similar to the way that system memory can be imaged from a firewire device. It is possible that the USB interface can also be used for imaging RAM. To the best of my knowledge no one has ever built such a device, which seems a missed opportunity given that there are ATA interfaces on most motherboards. *Lesson: many technical options remain unexplored.*

3.2.5. Drive failures produce better data

Many of the drives I purchased were dead-on-arrival. Others could only partially be imaged. I learned of *ddrescue* (Diaz, 2012) would read as far as possible going from the first sector of the disk to the last sector, then, upon encountering errors, would jump to last sector of the hard drive and then repeatedly skip toward the front of the drive, read a few sectors, and repeat. This algorithm works for a single bad spot on the hard drive, but it doesn’t work if there are multiple errors. I developed a disk imaging program called *aimage* which implemented a variety of recovery algorithms, such as attempting to repeatedly reread the problematic section; randomly seeking and reading; jumping ahead a few hundred kilobytes at each error, and reading from the last sector toward the first.

It appears that those who were selling broken drives on eBay didn’t bother to sanitize them. I once bought a 24-drive RAID array on eBay. Most of the drives had been wiped, but several of the drives had been swapped out of the RAID and had bad sectors. *Lesson: Drives with some bad sectors invariably have more sensitive information on them than drives that were in working condition when they were decommissioned.*

I found it difficult to maintain *aimage* because of frequent changes to the low-level Linux and FreeBSD I/O subsystems that the program required. As a result I abandoned *aimage*. Those looking for an AFF-aware imaging tool should use *FTKimager* (Access Data, 2011) or *guymanager* (Voncken, 2012). *Lesson: do research, and only to maintain software that implements a particular function when no other software is available.*

3.2.6. Numbering and naming

Initially I gave each disk image a unique number. Occasionally I made a mistake and gave the same number to more than one disk image, causing confusion. I experimented with using a randomly generated 128-bit number, with the rationale that no such number would ever be randomly generated twice. I learned that is annoying to work with file names containing 32 random hexadecimal numbers and abandoned this approach. *Lesson: Names must be short enough to be usable but long enough to be distinct.*

When I started acquiring data outside the US I discovered that the country of origin was the most important characteristic of a disk image. I adopted a naming scheme in which the first two characters are the ISO country code, followed by a two-digit batch number, a dash, and a four digit item number. (For example, CN07-0045 is the 45th disk of the 7th batch acquired from China.) Assigning a batch number allows different individuals in the same country to assign their own numbers. *Lesson: although it is advantageous to have names that contain no semantic content, it is significantly easier to work with names that have some semantic meaning.*

3.2.7. Path names

With thousands of disk images, tens of thousands of E01 files, and roughly a million individual files downloaded from US Government web servers, the full corpus has grown to roughly 30TB (Table 1). Very few systems in my organization have sufficient storage to hold the entire corpus. Fortunately, most users do not need access to the entire corpus in order to get their work done. Students copy to their personal workstations only the portions of the corpus that are relevant to their research.

To help structure the corpus we have implemented a directory structure that is the same for every machine that has a copy of the corpus. Originally we organized our files according to modality, legal status, and origin, in that order; recently I reorganized the corpus so that the top-level directory is origin and legal status, followed by modality. This has made it significantly easier to implement appropriate security and access controls, as researchers that have not had appropriate IRB training are not allowed access to the directory containing human subject data (Table 2)—and indeed, they can get most of their work done solely with */corp/nps/*. The placement of the M57-Patents scenario was complicated by the fact that the scenario contained disk images, memory dumps, and network packet captures: in some cases it is useful to have the materials grouped together, in others it is useful to have them grouped with their modality. I finally decided to place each modality with other files of the same type, and to create a *scenarios* hierarchy containing symbolic links pointing elsewhere. *Lesson: place access-control information as near to the root of a path name as possible.*

3.2.8. Anti-virus and indexing

The forensic corpora have caused unexpected problems on computers that periodically scan all files for viruses and for full-text indexing. Because the corpora contains large amounts of viruses and damaged files, we have seen them cause scanners to either crash or perform improperly. For example, Apple's "Spotlight" search engine repeatedly crashed when attempting to scan a directory and then proceeded to attempt re-scans. The result was excessive CPU load and, in one case, an index file that grew without bound until the indexing was disabled.

Lesson: Configure anti-virus scanners and other indexing tools (e.g., Apple's build_hd_index (Apple Computer, 2012)) to ignore directories that might contain raw forensic data.

Table 1
Current size of various forensic corpora.

Corpus Subset	Comp. Size	Files
Real Data Corpus:	29,000 GB	2394
GOVDOCS1 Corpus	490 GB	987,283
M57-Patents Scenario		
drives	417 GB	83
RAM	43 GB	89
net	4 GB	49

Table 2
Sample paths in the corpora.

Path	Contents
/corp/	Top-level directory for corpus
/corp/nist/	Data from NIST, including the NSRL
/corp/nps/	Data produced at NPS
/corp/nps/drives/	Disk images manufactured at NPS
/corp/nps/files/	Individual files produced at NPS
/corp/nps/files/govdocs1m	The Million Document Corpus
/corp/nps/files/govdocs1m /123/123456.jpg	File number 123456 in the million document corpus.
/corp/nps/malware/	Malware acquired by NPS
/corp/nps/malware/windows	Malware for Windows
/corp/nps/packets/	Network packets produced at NPS
/corp/nps/scenarios/	Worked scenarios containing disk images, packets, RAM, and ancillary materials
/corp/nus/	The Non-US Real Data Corpus
/corp/us/	The US Persons Real Data Corpus (not present on NPS machines)

3.2.9. Distribution and updates

There is no good way to distribute a 30TB data set. One approach is to manually split the data set up, put it on separate drives (we use internal SATA drives with docks) and ship them. This is our preferred method to give other organizations snapshots of the corpus, and I am developing an offline synchronization system that uses terabyte-sized external drives.

We have tried using *rsync* (Tridgell and Mackerras, 1996), but network connections are rarely fast enough. I am investigating efficient ways to send terabyte-sized files over gigabit Internet links with UDP-based file transfer protocols, which appear to have better utilization than TCP-based protocols. While scientists who work in the areas of high-energy physics and astronomy have similar bandwidth requirements, they typically have larger budgets, bigger teams, better Internet connections, longer time horizons, and data that are more-or-less uniform.

Lesson: solutions developed by other disciplines for distributing large files rarely work well when applied to DF without substantial reworking.

3.3. Corpus management–policy issues

Policy issues frequently arise when working with collections of sensitive data.

3.3.1. Privacy issues

In 1988 the US Supreme Court held that there is no privacy interest in trash (US Supreme Court, 1988). It seems reasonable to extend this ruling to hold that there is no privacy interest in the contents of data carrying devices that are sold on the secondary market. A similar conclusion also arises from a straightforward application of the "First Sale" doctrine (US Supreme Court, 1908) in copyright law. Although it would be inappropriate and probably immoral for me to release the contents of the drives that I purchase on the secondary market, I do not believe that it would be illegal to do so. In practice, I treat the information in the corpus as if it is private and confidential. *Lesson: just because something is legal, you may wish to think twice before you do it.*

3.3.2. Illegal content—financial, passwords, and copyright

The vast majority of media that we encounter have some kind of illegal content, typically illegally copied music. The criminal offense provisions of the US Copyright Act (17 USC §506) only apply to a person who "willfully infringes a copyright…for purposes of commercial advantage or private financial gain." This clearly does not apply to me, as I am not making money on the corpus. *Lesson: never sell access to DF data, even if you have personal ownership.*

It is also my assertion that assembling and distributing the corpus itself is "Fair Use" under 17 USC §107. US law defines a four-part test to determine whether or not Fair Use holds. The test considers 1) the purpose and character of the use (including whether the use is commercial or for non-profit educational purposes); 2) the nature of the copyrighted work; 3) the amount of the work that is copied; 4) the impact of the use on the market value for the copyrighted work. While the corpus clearly contains copyrighted data, I believe that my use of it for scientific research is clearly covered under Fair Use. *Lesson: understand Copyright Law before copying other people's data.*

Many media in the corpus also contain passwords and credit cards, both considered "access devices" under US law. The Computer Fraud and Abuse Act (18 USC §1029) states that anyone who "knowingly and with intent to defraud possesses fifteen or more devices which are counterfeit or unauthorized access devices" is committing a crime. I believe that the operative word here is *intent*. Clearly, my intent is research, not fraud. *Lesson: make sure your intent is scientific research, not fraud, so that any collection of access devices you create does not constitute criminal activity.*

3.3.3. Illegal content—pornography

In the US it is illegal to expose minors to pornography. On the other hand, pornography is widely distributed on the Internet and many seized computer systems contain pornography. It is thus likely that there is pornography in my disk corpus. *Lesson: do not give minors access to real DF data; do not intentionally extract pornography from research corpora.*

Child pornography presents a special problem. In general only law enforcement organizations may knowingly posses child pornography, and then only in conjunction with a criminal investigation. In two instances I have found file names that were highly suggestive of child pornography in the corpus:

1. In 2006 a disk was found with suggestive file names, but the file contents were overwritten with numerous copies of the movie *Monsters Inc.* Although I knew the source of the drive, the FBI chose not to investigate because no actual child pornography appeared to be present.
2. In 2011 one of my research affiliates discovered data on several drives with content and file names that were highly suggestive of sexual assaults against children. A government investigator determined individuals were not actually children. The investigation was terminated, since the images were apparently legal pornography that had been intentionally mislabeled.

Despite the outcome, in both cases I purged the disk images from the corpus and told my research partners to remove their copies as well. *Lesson: although there is no legal requirement to purge simulated child pornography from your corpus, its discovery will take up a lot of your time. It's better to get rid of data that may be incorrectly mistaken for child porn, rather than having to engage in lengthy explanations.*

3.3.4. Institutional Review Boards

Federally funded research in the US that involves human subjects or private data that is personally identifiable must be approved by an Institutional Review Board (IRB) holding an appropriate Federal assurance.

Some DF practitioners are confused to discover that federally funded research with used hard drives purchased on eBay, borrowed from students, or collected during the course of criminal investigations requires approval from an IRB. "We are clearly not working with human subjects," one researcher told me. He was wrong, as the so-called "Common Rule" clearly states:

45 CFR §46.102 Definitions.
f *Human subject* means a living individual about whom an investigator (whether professional or student) conducting research obtains

1. Data through intervention or interaction with the individual, or
2. Identifiable private information.

I have written elsewhere about the growing involvement of IRBs in computer science research (Garfinkel and Cranor, 2010) and how the mission of IRBs is slowly expanding (Garfinkel, 2008). Here I write about strategies for avoiding IRB review and problems that I have encountered with IRBs at other institutions.

An improper way to avoid IRB oversight would be to publish the contents of the hard drives on the Internet. While doing so would make the data literally non-private, eliminating the IRB overview, it would not be moral or ethical to do so.

A better approach is to avoid doing "research," which is defined in §46.102(d) as "a systematic investigation, including research development, testing and evaluation, designed to develop or contribute to generalizable knowledge." For example, tool testing is not research, provided that the tools and the algorithms they contain are developed using artificially constructed or fake data.

Personally I find it useful to use real data for research, so I obtain IRB approval for my work and require my collaborators to either obtain IRB approval from their own institutions or to affirm that they will not be performing research with the data as defined under Federal Law. So far I have experienced two notable problems:

- An IRB at one institution issued an approval but prohibited the PI from sharing his application with me. The IRB said the application was proprietary university information. I resolved this case by refusing to provide the researcher with the data, since I didn't know what he was approved to do.
- In two cases IRBs at other universities have concluded that research with the Real Data Corpus is exempt under the Common Rule. Even though the IRBs were clearly wrong, I abided by their by ruling, as it is not my job to police other IRBs and the researchers promised to keep the data private and not redistribute them without prior approval.

Lesson: While IRBs exist to protect human subjects, many have expanded their role to protect institutions and experimenters. Unfortunately this expanded role occasionally decreases the protection afforded human subjects. And even with the IRB watching over you, it's important to watch your back.

4. Lessons learned developing DF tools

This section discusses software engineering and design issues we have encountered while developing DF tools.

4.1. Platform and language

While Windows is clearly the dominant platform used by computer forensics practitioners, Linux and MacOS seem to be the dominant platforms used by forensics researchers. I have found that I cannot mandate a platform and instead need to deliver software that can be used on all three.

The easiest way to write multi-platform tools is to write command-line programs in C, C++, C#, Java or Python, as programs written in these languages can easily transfer between the three platforms. Although C has historically been the DF developer's language of choice, we have shifted to C++ so that we can use the STL collection and container classes. We have also been able to significantly improve the resistance of our programs to corrupt input data by putting all bounds and error checking in the C++ accessor methods so that they are systematically applied to all data accesses. The penalty for such checks is negligible on modern hardware.

Java has a reputation of running significantly slower than C/C++. Testing so far indicates that this reputation was only partially deserved. I created parallel implementations of several programs in C++ and Java including the NPS Bloom filter implementation, a hash-based carving prototype, and an early version of *bulk_extractor*. I measured the OpenSSL MD5 implementation and found that it was three times faster than Java's built-in MD5 implementation; I never published this work because I never finished the project. My goal was to replace Java's built-in MD5 implementation with OpenSSL called through the JNI or JNA interfaces. It might be faster, but I don't know.

Bruce Allen and I translated an early single-threaded version of *bulk_extractor* into Java using JFlex (Klein, 2009) and found that it ran three times faster than the single-threaded C++ implementation. This may have because the Java JIT re-optimizes object code during execution, or it may that Java was performing memory management in another thread, gaining some parallelism. The difference is relevant, because if the improved performance was due to opportunistic multi-threading in the Java's memory system, that advantage would be lost against the multi-threaded C++ *bulk_extractor*. This is an area that requires additional research.

While it is easy to write programs in Python, experience to date has shown that these programs are slow and memory-intensive. This is not a problem for programs that process evidence the size of files or memory dumps, or for most programs that process DFXML files, but it is a problem for processing entire disk images.

Lately we have been trying to move new software development to C#, as the CLR runs C# nearly as fast as C++ on Windows and C# has improved safety and type checking. Unfortunately there is only one implementation of C# for Mac and Linux, the implementation is several years behind Microsoft's, and its future is uncertain. So while we are exploring C#, we continue to use C++ for the majority of our development.

4.2. Parallelism and high performance computing

The data scale problem has forced me to spend a significant amount of effort on "plumbing" issues such multi-threading and high performance computing in an effort to squeeze additional performance. So far our efforts are mixed. In 2009 and 2010 my group spent a substantial effort developing MD5, SHA1 and AES implementations that could process data at many gigabytes per second on an IBM Cell Broadband Engine blade system (Dinolt et al., 2010), only to discover that the 10-Gig interface module would not be supported. A similar project at another school developed fast hash implementations on GPU co-processors: that project was terminated when the group discovered that the I/O bottleneck on modern GPUs was so slow that it was faster to hash on the host processor. On the other hand, our efforts at making *bulk_extractor* multi-threaded have been staggeringly successful.

4.3. All-in-one tools vs. single-use tools

Because there are many different kinds of forensic investigations, the same tool frequently needs to be applied to the same kind of data but for different purposes. One examiner might need to extract the visible text from a Microsoft Word file, while another might want the deleted text, a third might want residual metadata that indicates the document was edited on multiple computers, and a fourth might want proof as to whether or not the file was modified using a hex editor. Such wildly different use cases significantly complicate the task of tool development, documentation, and training. My experience argues that it is better to have a single tool than many:

- If there are many tools, most investigators will want to have them all. Splitting functionality into multiple tools complicates tool management without providing any real benefit to practitioners.

- Much of what a DF tools does—data ingest, decoding and enumerating data structures, preparing a report—is required no matter what kind of output is desired.
- There is a finite cost to packaging, distributing, and promoting a tool. When a tool has many functions this cost is amortized across a wider base.

One way to address the problem of different use cases is to have tools organize output into different sections or files, with one section providing information that is useful for typical cases, and another containing all of the extracted data. Outputs should ideally be both human and machine readable.

In the case of the Microsoft Word decoder example above, it would be possible to structure the tool's output such that it starts with a section that contains the Word file's text, followed by a section with the metadata and deleted text, and a final section containing all decoded internal structures in XML. Most forensic users will just refer to the text or the metadata, some will read the final section, and a few will write software to process it the final section as part of another tool chain. The XML could even be embedded in a PDF report as an attachment.

4.4. Evidence container file formats

Because of the diversity of tools and the general lack of user training, forensic software should be able to process inputs in any format. In practice a single input layer should allow tools to transparently handle disk images in raw, split-raw, EnCase or AFF formats. SleuthKit's *img* layer provides this capability but is not widely used for this purpose. (I don't use it in *bulk_extractor*, for example, due to usability problems.) AFFLIB provided an abstract facility to read both disk images and metadata, but I put AFFLIB3 into maintenance mode after the AFF4 announcement (Cohen et al., 2009); sadly, the production release of AFF4 has been delayed, and it remains unclear if AFF4 will read AFF3, raw, split-raw, and EnCase file formats.

Instead, I created a C++ iterator that allows bulk_extractor to read disk images in raw, split-raw, EnCase and AFF formats. The iterator is not sufficiently general for others to use, but it may be in the future.

With network packets the situation is better, with pcap being the universal format. Since taking over the tcpflow project (Elson and Garfinkel, 2011) I have modified the program to output flow data in DFXML format (§4.5). This summer it should be further modified to product data in a binary netflow format.

4.5. DFXML metadata and provenance

As I continued development of DF tools, I repeatedly encountered the need to represent data that was complex, highly structured, and frequently incomplete. For example, for a project on file carving, I wanted to be able to represent the number of fragments and the fragments' physical position for each fragment in a file. For another project on carving, I wanted to be able to represent metadata extracted from those fragments. For a feature extractor, I wanted to be able to represent files containing features.

The original approach of most DF tool developers was to create a separate file format for each of these tasks. For example, SleuthKit has a "body file" format that stores some kinds of metadata. This approach had the advantage of being fairly easy to implement and reasonably efficient for a single tool, but had numerous disadvantages:

- Because each output file is designed for a specific task, every program produces an output file with slightly different structures.
- Every program that wants to read the file needs to implement its own parser.
- Minor changes to the file format requires modifying every program that reads the files.

In addition to storing the results of forensic processes, I soon discovered that there was provenance information I wanted to store for each run of a program, including:

- The version of the program that was run.
- The computer on which the program had been compiled.
- The computer on which the program was run.
- When the program was run.
- The amount of CPU time that was required.
- The names of the input file.

The obvious way to address the scaling problem and to store the additional information was to use some kind of tagged file format. This would allow me both to store arbitrary name/value pairs and to tag any name or value with own set of name/value pairs. Thankfully I realized that this was the key insight of XML. I developed a new XML language called Digital Forensics XML (DFXML) which is specifically designed to represent the results of forensic processing, including all of the information in the SleuthKit body file and all of the provenance information mentioned above. Additional details can be found in (Garfinkel 2012).

Some practitioners have criticized my decision to use XML, arguing that other representations are superior. I disagree. Well-formatted XML can be read by both humans and software, and the overhead of XML is rarely material in the forensic context.

It turned out to be remarkably easy to get the developers of open source DF tools to support DFXML: I simply wrote the patches myself and provided them. Grenier was quite gracious in taking the patches for Photorec (Grenier, 2011), and Kornblum was gracious to take them for md5deep (Kornblum, 2011). I am working to have the format adopted by other open source tools and hope that commercial vendors will follow.

5. Related work

There have been several efforts to share DF lessons learned. Casey (2002) presented "practical lessons" in confronting encryption during the course investigations. More recently, Kim et al. (2009) shared lessons learned in creating a reference data set of Korean-language software.

Walls et al. (2011b) shares the authors' experience in developing digital forensic tools for use by law enforcement organizations. The authors explain that while success in DF is "strongly driven by practitioners who can readily adapt cutting-edge research," a variety of systematic barriers challenge these practitioners.

Harrison (2002) proposed creating a "Lessons Learned Repository." That effort and others have met with resistance from law enforcement practitioners, many of whom feel that such a repository could be used by defense attorneys to discredit examiners.

6. Conclusion

Digital Forensics is an exciting area in which to work, but it is exceedingly difficult because of the diversity of data that needs to be analyzed, the size of the data sets, and the mismatch between the technical skills of users and the difficulty of the work. These problems are likely to get worse over time, and our only way to survive the coming crisis is to concentrate on the development of new techniques that leverage our advantage—the ability to collect and maintain large data sets of other people's information. My research corpora are analogous to the kind of data that is acquired during the course of operations by law enforcement and the military; in building and maintaining this corpus I have encountered many problems that are increasingly relevant to others in the field. This paper describes some of the lessons that I have learned in the course my research in this area.

Acknowledgments

My introduction to computer programming was a "programmed instruction" book that I found at the Franklin Institute in Philadelphia's library. I was hooked. Sadly, in this age of Facebook and Cell phones it seems surprisingly harder for children to get exposure to programming at an early age. We must address this problem if we wish for our technological society to survive.

Beth Rosenberg and Joel Young read previous drafts of this article and provided useful comments. The feedback provided by the DFRWS anonymous reviewers was invaluable.

The views and opinions expressed in this document represent those of the authord and do not reflect those of the US Government or the Department of Defense. This document is a work of a US Government employee and as such is not subject to copyright.

References

Access Data. FTK imager; 2011.

Apple Computer. Apple Remote Desktop: how to disable build_hd_index; 2012.

Böhme R, Freiling FC, Gloe T, Kirchner M. Multimedia forensics is not computer forensics. In: IWCF'09: proceedings of the 3rd international workshop on computational forensics. Berlin, Heidelberg: Springer-Verlag; 2009. p. 90–103.

Casey E. Confronting encryption in computer investigations: practical lessons. In: Proceedings of the 2002 DFRWS conference; 2002.

Cohen MI, Garfinkel S, Schatz B. Extending the advanced forensic format to accommodate multiple data sources, logical evidence, arbitrary information and forensic workflow. In: Proceedings of DFRWS 2009. Montreal, Canada: Elsevier; 2009.

Diaz AD. GNU ddrescue manual; 2012.

Dinolt G, Allen B, Canright D, Garfinkel S. Parallelizing SHA-256, SHA-1 MD5 and AES on the cell Broadband engine. Technical Report NPS-CS-10-11. Monterey, CA: Naval Postgraduate School; 2010.

Elson J, Garfinkel S. Tcpflow; 2011.

Garfinkel S. Digital Forensics XML. Digital Investigation; 2012:8:161–174.

Garfinkel S, Cranor L. Institutional review boards and your research. Communications of the ACM; 2010:38–40.

Garfinkel S, Nelson A, White D, Roussev V. Using purpose-built functions and block hashes to enable small block and sub-file forensics. In: Proc. of the tenth annual DFRWS conference. Portland, OR: Elsevier; 2010.

Garfinkel S, Shelat A. Remembrance of data passed. IEEE Security and Privacy 2003;1(1):17–27.

Garfinkel SL. Design principles and patterns for computer systems that are simultaneously secure and usable. Ph.D. thesis; MIT; Cambridge, MA; 2005.

Garfinkel SL. IRBs and security research: myths, facts and mission creep. In: Usability, psychology and security 2008 (Co-located with the 5th USENIX symposium on Networked Systems Design and Implementation (NSDI'08)). San Francisco, CA: Usenix; 2008.

Garfinkel SL. Providing cryptographic security and evidentiary chain-of-custody with the advanced forensic format, library, and tools. The International Journal of Digital Crime and Forensics 2009;1:1–28.

Garfinkel SL, Farrell P, Roussev V, Dinolt G. Bringing science to digital forensics with standardized forensic corpora. In: Proceedings of the 9th Annual Digital Forensic Research Workshop (DFRWS). Quebec, CA: Elsevier; 2009.

Garfinkel SL, Malan DJ, Dubec KA, Stevens CC, Pham C. Disk imaging with the advanced forensic format, library and tools. In: Research advances in digital forensics (Second annual IFIP WG 11.9 international conference on digital forensics). Springer; 2006.

Grenier C. Photorec; 2011. http://www.cgsecurity.org/wiki/PhotoRec [accessed 03.12.11].

Guidance Software. Form 10-q; 2011.

Harrison W. A lessons learned repository for computer forensics. In: Proceedings of the 2002 DFRWS conference; 2002.

Hibshi H, Vidas T, Cranor L. Usability of forensics tools: a user study. In: IT Security Incident Management and IT Forensics (IMF), 2011 sixth international conference on. IEEE; 2011. p. 81–91.

Kim K, Park S, Chang T, Lee C, Baek S. Lessons learned from the construction of a Korean software reference data set for digital forensics. In: Proceedings of the 2009 DFRWS conference; 2009.

Klein G. Jflex: the fast scanner generator for java; 2009.

Kornblum J. md5deep and hashdeep—latest version 4.1; 2011. http://md5deep.sourceforge.net/ [accessed 18.02.12].

Lawson TF. Before the verdict and beyond the verdict: the CSI infection within modern criminal jury trials. Loyola University Chicago Law Journal 2009;41(1).

Metz J. libewf: project info; 2011. http://sourceforge.net/projects/libewf/ [accessed 03.12.11].

Nelson AJ. RegXML: XML conversion of the Windows Registry for forensic processing and distribution. In: Chow KP, Shenoi S, editors. Advances in digital forensics VIII, in press. Springer; IFIP Advances in Information and Communication Technology; 2012.

Reynolds JK, Postel J. RFC 1000: request for comments reference guide; 1987. Obsoletes RFC0999.

Shelby RC. September 11 and the imperative of reform in the U.S. Intelligence Community; 2002. Additional Views of Senator Richard C. Shelby Vice Chairman, Senate Select Committee on Intelligence.

Shelton DE. The 'CSI Effect': does it really exist? NIJ Journal 2008;259.

Snow RL. Technology and law enforcement: from gumshoe to gamma rays. Greenwood Publishing Group; 2007.

Tridgell A, Mackerras P. The rsync algorithm. Technical Report TR-CS-96-05; ANU Computer Science Technical Reports; 1996.

US Supreme Court. Bobbs-Merrill Co. v. Straus; 1908. 210 US 339.

US Supreme Court. California v. Greenwood; 1988. 486 US 35.

Voncken G. Guymanager; 2012. http://guymager.sourceforge.net/.

Walls RJ, Learned-Miller E, Levine BN. Forensic triage for mobile phones with dec0de. In: Proceedings of the 20th USENIX security symposium. USENIX; 2011a.

Walls RJ, Levine BN, Liberatore M, Shields C. Effective digital forensics research is investigator-centric. In: Proc. USENIX workshop on Hot Topics in Security (HotSec); 2011b.

Wright C, Kleiman D, Shyaam Sundhar RS. Overwriting hard drive data: the great wiping controversy. In: Lecture notes in computer science/ICISS 2008. Springer; 2008. p. 243–57.