Available online at www.sciencedirect.com

SciVerse ScienceDirect

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

Computers & Security

Digital media triage with bulk data analysis and *bulk_extractor*

*Simson L. Garfinkel**

Naval Postgraduate School, 900N Glebe, Arlington, VA 22207, USA

ARTICLE INFO

Article history:
Received 8 December 2011
Received in revised form
4 July 2012
Accepted 14 September 2012

Keywords:
Digital forensics
Bulk data analysis
bulk_extractor
Optimistic decompression
Windows hibernation files
EnCase
Forensic path
Margin
Parallelized forensic analysis
Stream-based forensics

ABSTRACT

Bulk data analysis eschews file extraction and analysis, common in forensic practice today, and instead processes data in "bulk," recognizing and extracting salient details ("features") of use in the typical digital forensics investigation. This article presents the requirements, design and implementation of the *bulk_extractor*, a high-performance carving and feature extraction tool that uses bulk data analysis to allow the triage and rapid exploitation of digital media. Bulk data analysis and the *bulk_extractor* are designed to complement traditional forensic approaches, not replace them. The approach and implementation offer several important advances over today's forensic tools, including optimistic decompression of compressed data, context-based stop-lists, and the use of a "forensic path" to document both the physical location and forensic transformations necessary to reconstruct extracted evidence. The *bulk_extractor* is a stream-based forensic tool, meaning that it scans the entire media from beginning to end without seeking the disk head, and is fully parallelized, allowing it to work at the maximum I/O capabilities of the underlying hardware (provided that the system has sufficient CPU resources). Although *bulk_extractor* was developed as a research prototype, it has proved useful in actual police investigations, two of which this article recounts.

Published by Elsevier Ltd.

1. Introduction

Digital forensics investigations have grown more difficult as the capacity and diversity of devices containing digital evidence increases.

Many approaches have been proposed for addressing the data onslaught, including parallelization and multiprocessing (Ayers, 2009; Richard and Roussev, 2006), statistical sampling (Garfinkel et al., 2010; Mora, 2010), and even extending the time a suspect can be held without charge so that evidence may be analyzed (Secretary of State for the Home Department, 2007). But no matter what approach is used to improve performance, so long as a backlog exists, some process must allocate limited forensic resources.

Diversity of media represents a different kind of challenge. Increasingly evidence needs to be analyzed that is in file system and file formats not supported by current tools. Frequently the only way to analyze such data is to scavenge it for printable strings and rely on the human examiner to make sense of the data that are manually recovered.

Many organizations prioritize forensic processing with reference to the nature of the case, without taking into account the contents of the media itself. For example, the FBI's regional computer forensics laboratories give media

* Tel.: +1 831 656 3389.
E-mail address: slgarfin@nps.edu.

0167-4048/$ – see front matter Published by Elsevier Ltd.
http://dx.doi.org/10.1016/j.cose.2012.09.011

from terrorism cases highest priority (RCF Laboratory, 2008). Many local police departments use trial dates and the statute of limitations to prioritize their workload.

Triage is a term widely used to denote the prioritization of work according to a quality inherent in the objects being acted upon. This article proposes expanding triage to include the results of a rapid and largely automated analysis, performed when the media is first encountered. The results of such triage can frequently reveal if the media is likely to have intelligence value and, therefore, if the media should be prioritized for immediate deep analysis.

This article describes advances recently made in *bulk data analysis*, an approach for performing digital forensics that eschews file extraction, and instead focuses on the processing of bulk data and the extraction of salient details ("features"). Unlike file-based approaches, bulk data analysis is particularly well-suited to triage. Furthermore, combining forensic feature extraction (Garfinkel, 2006) with optimistic decompression (§3.3) allows the recovery of features that are otherwise missed using current tools. Triage using bulk data analysis is particularly useful in cases that involve multiple pieces of digital evidence. For example, in a case involving credit card theft, bulk data analysis can quickly identify which laptop or cell phone has the largest collection of credit card numbers, allowing investigators to prioritize. In an organized crime or public corruption case, bulk data analysis can rapidly identify which hard drives or cell phones contain email addresses of interest and rapidly generate new leads. Users have seen that the specialized bulk analysis software presented in this article can frequently process digital media significantly faster than today's industry standard tools. Thus, systematic bulk data analysis has the potential for transforming digital media analysis in law enforcement from a conviction-support tool (through the identification of illegal materials) to a tool that provides support early in an investigation.

1.1. *Contributions*

This paper makes multiple contributions to the theory and practice of digital forensics. First, it shows that bulk data analysis without reference to an underlying file system can be productively used as a fast analysis step during the early part of an investigation. It shows that bulk data analysis is easily adopted to stream processing and multi-threading. It shows that recall rates of bulk data analysis can be significantly improved through the use of optimistic decompression and recursive re-analysis, for the simple reason that many features of forensic interest are present in compressed byte streams located in the unallocated regions of typical file systems.

This paper presents *bulk_extractor*, a powerful tool for performing bulk data analysis. It shares the experience of developing *bulk_extractor*, providing a model for the development of future digital forensic tools.

This paper introduces the concept of the context-sensitive stop list, and explains how stop lists that do not include context are susceptible to manipulation by criminals and other adversaries.

This paper presents a new forensic disk image designed for testing modern digital forensic tools. It presents an evaluation using this image that compares *bulk_extractor* with Guidance Software's EnCase (Guidance Software, Inc., 2011) and the traditional approach of running the Unix *strings* and *grep* commands. It shows that *bulk_extractor* can recover more kinds of high-value forensic features than current tools and explains the reasons for improved recall performance.

Finally, this paper presents two real-world case studies that show how features provided by bulk data analysis have been used productively in forensic investigations.

1.2. *Outline of this article*

This concludes the introduction. Section 2 discusses related work. Section 3 discusses the design of the bulk analysis system while Section 4 discusses the *bulk_extractor* implementation. Section 5 presents the results of *bulk_extractor* run against both test data and hundreds of forensic images. Section 6 presents two case studies. Section 7 discusses limitations of our architecture and opportunities for future work. Section 8 concludes.

2. Related work

2.1. *File-based forensics, bulk data analysis, and file carving*

Two complementary approaches are now used in the processing of digital evidence: *file-based approaches* and *bulk data analysis*.

File-based approaches are widely used by digital forensic examiners and implemented by popular tools such as Guidance Software's EnCase (Guidance Software, Inc., 2011) and AccessData's FTK (AccessData, 2011). Such tools operate by finding, extracting, identifying and processing *files*—that is, a sequential collection of bytes pointed to by file system metadata.

Because they mirror the way that users interact with computers, file-based approaches have the advantage of being easy to understand. File-based approaches also integrate well with most legal systems, as extracted files can be printed and entered into evidence. They have the disadvantage of ignoring data not contained within files. Additional limitations of file-based approaches are discussed elsewhere (Garfinkel, 2010).

In *bulk data analysis*, digital content is examined without regard to file system metadata. Instead, data of interest is identified by content and processed, extracted, and reported as necessary. *File carving* is an example of bulk data analysis, although it is limited, because file carving ignores bulk data that cannot be assembled into files.

File-based approaches and bulk data analysis are complementary. In file-based approaches the access of the forensic tool mirrors that of the media's original user and of typical computer users. This makes the results easier to put in context, easier to explain to those who may not have a technical background, and potentially easier to validate with extrinsic data. Bulk data approaches have the advantage of being applicable to all types of computer systems, file systems, and file types—especially those not handled by existing tools. Bulk data analysis can also be readily applied to media that has been damaged or partially overwritten. Finally, in many cases bulk data analysis can find information that is not recoverable through file-based approaches because of limitations in specific file-based decoders.

2.2. *Triage*

Rogers et al. (2006) presented a Field Triage Process Model that focused on approaches for triage using existing tools. Asserting that "the value of planning and pre-raid intelligence cannot be over-emphasized," the authors presented a methodical and largely manual triage process that involved first ranking the media to be analyzed and then performing a systematic exploration of the data that each device contained.

Automated tools with limited triage functions are available commercially. For example, ADF Triage by ADF Solutions can scan a subject hard drive for documents containing specific keywords and images likely to be pornographic (Parsonage, 2009).

Pearson and Watson (2010) published the book *Digital Triage* which is heavily based on the experience of Explosive Ordinance Disposal (EOD) teams in Iraq, with chapters on battlefield forensics, the conducting of pre- and post-blast investigations, and the analysis of cell phones on the battlefield.

The work presented in this paper is most similar to the DEC0DE system developed by Walls et al. (2011). That system uses data-driven dynamic programming to identify call logs and address book entries in cell phone memory dumps without advance knowledge of how a vendor formats the information. The system uses block hash filtering to find data blocks common with other phones; these blocks are then removed, leaving blocks of data likely to contain information specific to the subject phone. This resulting data is analyzed with a probabilistic state machine. The authors state that their approach can process a 64 MB phone image in 15 min. The approach embodied in DEC0DE is not applicable to digital forensics in general, but is restricted to the limited domain of extracting structured information, such as extracting call detail records and SMS logs from cell phone memory dumps.

Clearly, even without specialized tools, examiners will engage in workflow prioritization whenever there exists a backlog of media to process. Nevertheless, policy and experience indicates that frequently there is no triage process at all, as work is frequently prioritized without reference to the data contained in the media.

2.3. *Recursive re-analysis and the recovery of compressed information*

The approach of recursively re-analyzing data during forensic analysis is widely used by anti-virus programs, existing media forensics tools (*e.g.* Guidance Software, Inc., 2011; AccessData, 2011), and network forensics tools. However, these tools generally apply re-analysis solely to intact container files. PyFlag (Cohen, 2008) applies recursive re-analysis to data in runs of unallocated sectors, but only attempts to decompress from the beginning of each run. The NetIntercept (Corey et al., 2002) network forensic tool automatically decompresses compressed network streams, but only for protocols known to use compression.

2.4. *Single-block forensics*

Prior research introduced a simplified form of bulk data processing that performed analysis of single sectors (Garfinkel et al., 2010). The approach presented here is more comprehensive, as it will identify features that cross sectors or span multiple sectors. Further, this work also includes recursive re-analysis of bulk data, while the prior single-block work did not.

2.5. *Forensic feature extraction*

Computer media frequently contains a persistent record of a subject's associates, activities, and financial activities. As such, software that automatically searches for these types of artifacts can aid many investigations. The term *forensic feature extraction* describes this process (Garfinkel, 2006).

Many forensic tools allow searching files and unallocated sectors for strings that match user-specified regular expressions. Vendors and trainers publish lists of regular expressions that match email addresses, US telephone numbers, US social security numbers, credit card numbers, IP addresses, and other kinds of information typically useful in an investigation (*e.g.* AccessData Corporation, 2008; Bunting, 2008, p. 304–305). Credit card number searches are also performed by the Cornell University Spider forensic tool (Cornell University IT Security Office, 2008). The University of Michigan Information Technology Security Services reviewed several tools for discovering credit card and social security numbers in 2008.

The work presented here improves prior work in forensic feature extraction by using hand-tuned rules that consider local context, improving the precision of extraction without adversely impacting recall. Furthermore, existing feature extraction systems do not in general extract features from compressed or otherwise encoded data.

2.6. *Parallel processing*

More than four decades of research has resulted in numerous approaches for speeding text processing through the use of parallelism. Indeed, Bird et al. discussed parallelized searching of a 50 GB database using specialized machines in 1977. Similar efforts continue to this day (Konstantopoulos et al., 2009).

Although there have been some attempts to parallelize forensic tools, little of this work is being used in the field. For example, Marziale et al. adapted the Scalpel file carver to offload complex computations to a GPU, but that version of Scalpel was never released. Collange et al. (2009) considered the use of GPUs to speed hash-based carving, but no commercial forensic tool uses GPUs for hashing due to I/O limitations. Urias et al. (2008) considered a variety of issues involving the parallelized processing of RAID storage systems, but despite increasing media sizes, tools such as EnCase and FTK are still largely single-threaded.

The parallelization described in this paper is distinguished from prior efforts in that it is easily implemented and automatically takes advantage of general-purpose multi-core CPUs, the most common form of parallelized hardware available. As such, this parallelism can be readily used by current practitioners on hardware that they already have. Indeed, the thread pool implementation described in this paper was recently incorporated into the popular *md5deep* hashing program (Kornblum, 2011).

2.7. *Tool validation*

The US Supreme Court held in *Daubert* that scientific evidence presented in court must involve established techniques that are peer-reviewed and have measurable error rates (Daubert v. Merrell Dow Pharmaceuticals, 509 US 579, 1993; Committee on Identifying the Needs of the Forensic Science Community, 2009). But little is known about the accuracy of feature extraction and string search of today's forensic tools (Lyle, 2010). The National Institute of Standards and Technology's Computer Forensics Tool Testing Program created draft specifications for String Search and Deleted File Recovery (CFTT, 2008), but the drafts have not advanced past an initial version and no test results have been published. Important real-world requirements are missing from these drafts, such as the ability to recover information from complex document formats and compressed data.

In the absence of a standard, Guo et al. (2010) advocate testing tools with simplified data sets containing known targets—for example, by creating documents containing the word "evidence" and then searching for the word. This paper presents an improvement to Guo's procedure that involves embedding different targets in different file formats (§5.1). This improvement makes it significantly easier to determine the source of each extracted feature, since each feature is self-identifying. A similar approach has since been identified for use in the NIST Deleted File Recovery Test Images after the NIST team was provided with a draft of this paper (CFTT, 2012).

3. Introducing *bulk_extractor*

The *bulk_extractor* is a program that extracts email addresses, credit card numbers, URLs, and other types of information from any kind of digital evidence. The program operates on disk images in raw, split-raw, EnCase E01 (Metz, 2008), and AFF (Garfinkel et al., 2006) formats, but the program has also been used productively on sessionized TCP/IP traffic, memory dumps, and archives of files downloaded from the Internet. The program can also directly analyze media directly connected to the analyst's computer—for example, with a write blocker. The data to be analyzed are divided into *pages* that are separately processed by one or more *scanners*. Identified features are stored in *feature files*, a simple line-based format containing extracted features, their location, and local context.

The *bulk_extractor* detects and optimistically decompress data in ZIP, gzip, and Microsoft's XPress Block Memory Compression algorithm (Suiche, 2008). This has proven useful in recovering email addresses from within fragments of corrupted Windows hibernation files.

The *bulk_extractor* gets its speed through the use of GNU flex (The Flex Project, 2008), which allows multiple regular expressions to be compiled into a single finite state machine, and multi-threading (§4.4), which allows multiple pages to be analyzed at the same time on different cores.

After the features have been extracted, *bulk_extractor* builds a histogram of email addresses, Google search terms, and other extracted features. Stop lists can remove features not relevant to a case.

The remainder of this section introduces *bulk_extractor* with a typical case and then presents the program's overall design; the following section (§4) discusses the current implementation; Section 5 presents approaches for validation.

3.1. *Use case*

The *bulk_extractor* is designed to be used in the early part of an investigation involving digital media. A typical case might involve the analysis of 20 laptops and desktops seized from suspected members of a child exploitation group. Each piece of subject media is connected to an analyst workstation with a write-blocker and directly processed with *bulk_extractor*. (Time to initiate: approximately 5 min per machine.) The *bulk_extractor* runs in a batch, unattended operation. (Time to process: between 1 and 8 h per piece of media, depending on size and complexity of the subject data.)

Each running instance of *bulk_extractor* creates a directory where the program's output is stored. The output consists of one or more *feature files* (Fig. 2). Each feature file is a text file that contains the location of each feature found, the feature itself, and the feature surrounded by its local context (typically 16 characters from either side of the feature). Typical feature files are *email.txt* for email addresses, *url.txt* for URLs, *aes.txt* for AES keys, and so on. Some of the information that is present in the feature files originated in compressed data on the subject disk. For example, many URLs and email addresses were present in browser cache entries compressed with the gzip compression algorithm. Because it was compressed, the data would not be evident simply by running the Unix "strings" program or by a manual examination of the disk sectors.

When the extraction phase is finished, each instance of *bulk_extractor* reads the feature files and creates a feature histogram for each file. Post-processing also extracts a histogram of popular search terms.

When the program finishes, the examiner may manually review the histogram files:

- The list of email addresses provides the examiner with a quick report of individuals that may have some connection to the drive. Email addresses can appear on a drive for many reasons—they can be in email messages, or in the web cache from webmail, or in a web cache because they appeared at the bottom of a news article that's being read, or even because they were in a Microsoft Office document.

However, experience has shown that the most common email addresses on a drive are typically present because they were in multiple email or webmail messages, and that is usually because they are associated with either the drive's primary user or one of that person's correspondents.
- Search terms can be used as an indicator of the computer user's intent.
- The presence, frequency and number of credit card numbers can be used to infer if the drive had a large number of credit card numbers (an indication of either credit card processing or credit card fraud), or a high frequency of very few credit card numbers (an indicator of frequent e-commerce).

Tools such as "grep" can be used to scan the list of URLs to extract identifiers for Facebook, Microsoft Live, and other online services. Lists of identifiers can also be uploaded to other systems for correlation with law enforcement databases.

Although the feature files can be directly entered into evidence or used as the basis of a formal report, more commonly the examiner will use the data to inform a more detailed analysis of the media with a traditional forensic tool such as EnCase or FTK. In this manner, *bulk_extractor* is used for triage—that is, to prioritize analysis based on the content of the media itself, rather than the circumstances surrounding the media's capture.

Because it is easy to use and readily integrated with existing forensic processes, *bulk_extractor* has been adopted by a number of law enforcement organizations, and its use is growing.

3.2. *Requirements study*

Between 2003 and 2005 a prototype bulk media analysis tool was developed by the author to assist in an unrelated investigation. Experience showed the tool to be significantly faster than file-based systems and allowed easy answer to questions of interest early in an investigation, such as *Does this drive contain sensitive information? What search terms were used?* and *what is the most common email address on the drive?*

Because the prototype did not map to an existing forensic tool category, a series of unstructured interviews were held with local, state and federal law enforcement (LE) forensic examiners to determine if there was a need for this new kind of tool. In total, approximately 20 interviews took place between 2005 and 2008.

Although it may seem that the tool development described here was the result of documented needs stated by LE, this was not the case. LE practitioners interviewed at the time were generally pleased with their then-current tools, which seemed quite powerful and had required considerable effort to master. Examiners merely wanted their existing tools to run faster—they were not looking for tools that implemented fundamentally new approaches. Indeed, at the beginning of the interviews, several LE practitioners and trainers spoke derisively of the desire to create a so-called "get evidence" button. Such a button could not be created, these practitioners asserted, because a computer would never be able to make sense of all the information left behind on a digital storage device and arrange it in a manner that was consistent with the objectives of a case.

It was only after seeing some of the initial results of the early prototype that some analysts became enthusiastic about the work and requested that the tool be further developed to extract specific types of information, including:

- Email addresses
- Credit card numbers, including track 2 information
- Search terms (extracted from URLs)
- Phone numbers
- GPS coordinates
- EXIF (Exchangeable Image File Format) information from JPEG images
- A list of all words present on the disk, for use in password cracking

Interviewees also provided a number of operational requirements:

- Run on Windows, Linux and Macintosh-based systems
- Operate on raw disk images, split-raw volumes, EnCase E01 evidence containers, and AFF evidence containers
- Perform batch analysis with no user input
- Allow users to provide additional regular expressions for searches
- Automatically extract features from compressed data
- Run as fast as the physical drive or storage system could deliver data
- Identify the specific files in the file system that are the source of the extracted features
- Produce output as an easy to use text file
- Never crash

The interviews revealed that the primary need for such a tool was triage—to prioritize which pieces of digital evidence should be analyzed first, and to identify specific email addresses for follow-up investigation. Final analysis, however, would typically be performed with an "approved" tool.

3.3. *Forensic scanners, feature extractors and optimistic decompression*

The *bulk_extractor* employs multiple *scanners* that run sequentially on raw digital evidence. These scanners are provided with a buffer to analyze (initially corresponding to a 16 MiB page of data read from the disk image) the location or *path* of the buffer's first byte, and a mechanism for recording extracted features. Special logic is used to handle features that span across buffer borders (§4.3). All buffers are processed by all scanners until there are no more buffers to analyze. At this point the program performs post-processing and finally exits.

There are two types of scanners. *Basic scanners* are limited to analyzing the buffer and recording what they find. An example is the email scanner *scan_email*, which can find email addresses, RFC822 headers, and other recognizable strings likely to be in email messages.

Recursive scanners, as their name implies, can decode data and pass it back to the buffer processor for re-analysis. An example is *scan_zip*, which detects the components of ZIP files, records the ZIP header as an XML feature, decompresses the data, and passes the decompressed data back to the buffer processor. Most of *bulk_extractor*'s recursive scanners are optimistic. That is, they scan the entire buffer for data that can be decompressed or otherwise transformed and, if they find it, they transform it as appropriate. In addition to decompressing, *bulk_extractor* uses optimistic transformations for BASE64 decoding, PDF text extraction, and other encodings. Optimistic decoding produces significantly higher recall rates than approaches that only decode data from specifically recognized file formats.

The speed of the forensic tool is obviously impacted by the use of additional scanners: the degree of the impact depends on the data being analyzed. A disk image that contains no compressed data will be processed more slowly merely because the tool scans for compressed data; in testing this degradation is not significant. Significant amounts of compressed data, in contrast, will significantly slow processing, especially if compressed data is contained within other compressed regions.

Forensic programs that recursively process compressed data must guard against *decompression bombs*—files that, when fully decompressed, extend to many terabytes, petabytes, or more (Aera Network Security, 2009). The *bulk_extractor* implements three defenses against compression bombs. First, only a configurable portion of each compressed stream is decompressed. Second, the page processor will not call the recursive scanners when the depth reaches a configurable limit (by default, five recursions). Finally, the tool computes the cryptographic hash for each compressed region prior to decompression; regions that have the same hash are only decompressed once.

3.4. *Feature files*

Analysts requested that the tool provide output as a simple text file that could be viewed with an editor or processed with other "scripting" tools. Realizing this request is *bulk_extractor*'s *feature file format*, a tab-delimited text file containing the offset where each feature was found, the feature itself, and a configurable number of bytes that precede and follow the feature in the evidence file (Fig. 2). Feature files are not sorted but are loosely ordered. The order is "loose" because it is determined, in part, by the execution order of the multiple threads. As a result, running *bulk_extractor* twice on the same subject media will likely result in feature files that contain the same *lines*, but for which the lines appear in a different order. (Sorting the lines during processing would require either additional memory or stalling one or more threads, both unacceptable solutions.)

When it is necessary to report multiple values associated with each extracted feature, the second and/or third fields of the feature file can be replaced with an XML fragment. For example, the JPEG scanner uses a block of XML to report all of the fields associated with EXIF structures found within embedded JPEGs.

3.5. *Forensic location, path, and file system correlation*

For reporting purposes it is important to identify the location at which each piece of extracted information is found. This can be challenging when using tools that have the ability to extract information from within compressed objects, because it is also necessary to document how the data must be decompressed or otherwise decoded.

There are at least five potential sources of compressed data on a hard drive:

1. Many web browsers download data from web servers with gzip compression and persist the compressed stream directly to the web cache. (The percentage of web servers employing compression increased from less than 5%–30% between 2003 and 2010 (Port80 Software, 2010) because compression significantly increases web performance (Pierzchala, 2006; Srinivasan, 2003).)
2. NTFS file compression may result in disk sectors that contain compressed data. The most commonly compressed files are Windows restore points, as the operating system compresses these automatically. However, users may choose to have *any* file compressed.
3. Windows hibernation files frequently contain forensically important information. Complicating access to this file is Microsoft's use of a proprietary compression algorithm called Xpress (Suiche, 2008) and the fact that Windows overwrites the beginning of the hibernation file when the operating system resumes from hibernation. Also, the hibernation file's location on the hard drive moves as a result of NTFS defragmentation operations (Beverly et al., 2011); thus, any software that hopes to recover features from hibernation files must be able to decompress incomplete hibernation file fragments.
4. Files are increasingly bundled together and distributed as ZIP, RAR, or *.tar.gz* archives for convenience and to decrease bandwidth requirements. These files are frequently written to a hard drive. If deleted, one of the components may be overwritten while the others remain.
5. The *.docx* and *.pptx* file formats used by Microsoft Office store content as compressed XML files in ZIP archives (Garfinkel and Migletz, 2009).

Consider a message containing a set of credit card numbers viewed using a webmail service. If the web client and server both support HTTP compression, the web page will most likely be downloaded as a gzip-compressed stream; both Firefox and Internet Explorer will store the compressed stream in the browser cache. If the computer is suspended, the web browser' memory may be compressed with Xpress and stored in the hibernation file. It is not enough simply to report *where* the credit card numbers are found on the subject's disk, because looking at the disk sectors with a hex editor will not show human-readable strings: it is also necessary to explain *how* the data must be transformed to make them intelligible.

Current forensic tools do not encounter this problem because they ignore compressed data that is not in known compressed file container formats.

To resolve this problem, *bulk_extractor* reports a *forensic path* for each feature found. For features recovered from uncompressed data, the forensic path is simply the distance in bytes from the beginning of the media. In cases where the feature is contained within an object that is decompressed or otherwise processed by a recursive scanner, the forensic path contains information that can be used to repeat the decoding process.

For example, the fourth line of Fig. 2 indicates that jbarnes@virtuousgeek.org is found by decompressing the gzip stream found at byte offset 318,707,924 from the start of the disk image; the email address occurs 70 bytes from the start of the decompressed stream. (The email address is contained within the file */casper/filesystem.squashfs*.)

Forensic paths can be extended: the email address nelson@crynwr.com in Fig. 2 is found by decompressing the gzip stream that begins at byte offset 946,315,592; 6400 bytes into the decompressed stream is a second compressed stream; the email address is found 1600 bytes into that stream. (This email address appears within the file *gnash-common_0.8.4-0ubuntu1_i386.deb* in the directory */var/cache/apt/archives/*.)

Forensic paths can be readily translated to a specific location in a resident or deleted file with a file system map. The *fiwalk* program (Garfinkel, 2009) produces such a map in just a few minutes for most disk drives smaller than a terabyte; the operation is fast because only file system metadata is examined. The program *file_locations.py*, included with *bulk_extractor*, can then be used to annotate a feature file with the file names corresponding to the sectors from which the features were extracted.

To recap: today's practice for describing the location of a feature extracted from a disk image is to report the sector number or offset where the evidence is found. The evidence can then be examined with a tool such as a hex editor to verify the existence of the feature. This approach simply does not work when the feature resides within compressed data: examining the sector with a hex editor merely shows binary data. The forensic path, introduced here, provides a clear, concise and unambiguous way to describe both the location of the extracted features and the specific decoding operations that need to be executed in order to recover the data.

3.6. *Histogram processing*

Frequency distribution histograms can be of significant use in forensic investigations (Garfinkel, 2006). For example, a frequency histogram of email addresses found on a hard drive readily identifies the drive's primary user and that person's primary contacts.

Histogram generation is integrated with the feature recording system so that histograms can be created for any feature or feature substring at the conclusion of media processing. For example, the regular expression below extracts search terms provided to Google, Yahoo, and other popular

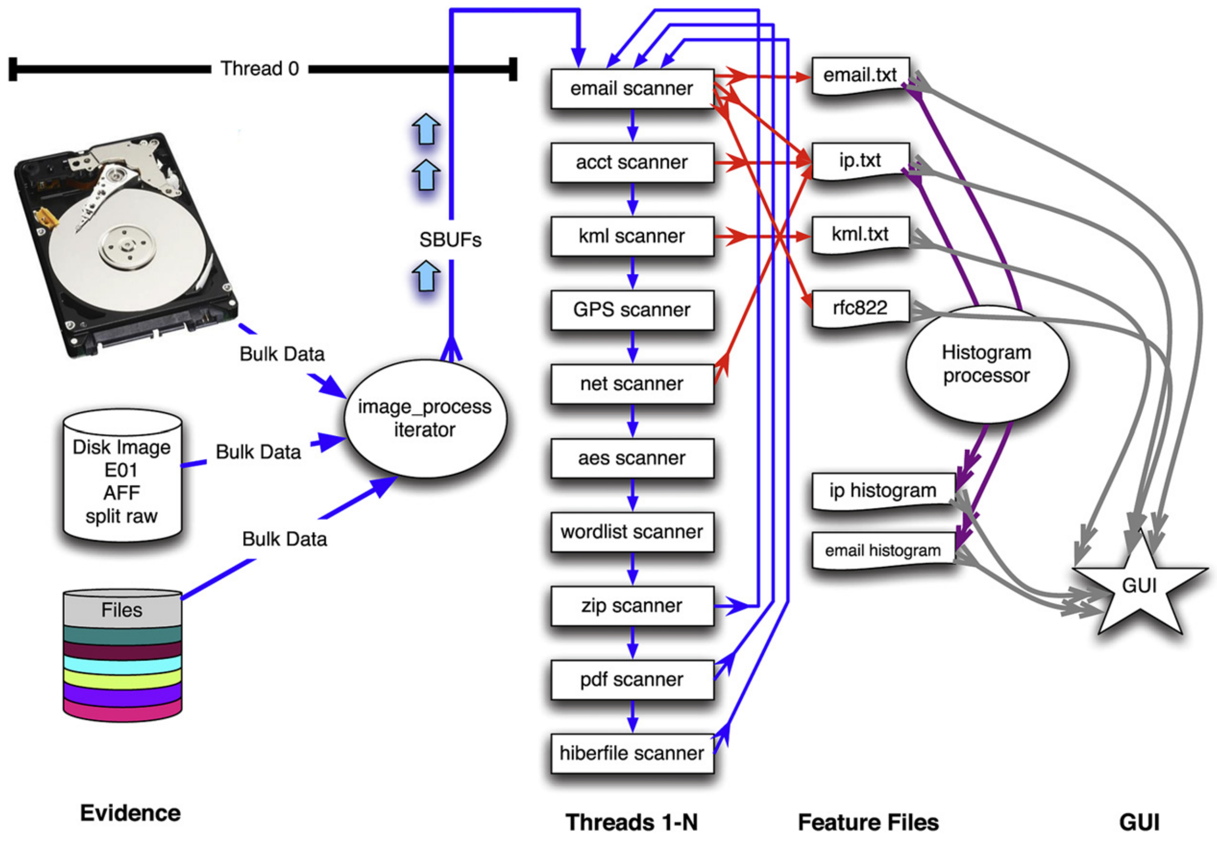

Fig. 1 – Diagram showing overview of the *bulk_extractor* architecture. Thread 0 reads data from a physical disk, disk image, or individual files and puts data into buffer structures called "sbufs". These buffers are processed sequentially by the scanners operating in the thread pool. Some of the scanners (*e.g.* zip, pdf and hiberfile) are recursive; they create new sbufs, which are in turn processed by all of the scanners. Features that are extracted are stored in feature files which are, in turn, processed by the histogram processor into histogram files. A graphical user interface (GUI), not described in this article, allows the resultant feature files to be browsed at the conclusion of the processing.

```
317697633         micke@imendio.com        kael Hallendal <micke@imendio.com>_Alexander Lars
317697671         alexl@redhat.com         xander Larsson <alexl@redhat.com>_Shaun McCance
317697704         shaunm@gnome.org         _Shaun McCance <shaunm@gnome.org>_3___x__][s_8__
318707924-GZIP-70  jbarnes@virtuousgeek.org  Jesse Barnes <jbarnes@virtuousgeek.org>_Date:   Wed
318707924-GZIP-647 jcristau@debian.org      Julien Cristau <jcristau@debian.org>_Date:   Wed Au
...
946315592-GZIP-64000-GZIP-1600  nelson@crynwr.com  Russell Nelson <nelson@crynwr.com>___[9976] ne
946315592-GZIP-64000-GZIP-16095 strk@keybit.net    androSantilli <strk@keybit.net>___[9880] Anoth
```

Fig. 2 – Two excerpts from a feature file generated by processing the disk image *ubnist1.gen3.E01* (Garfinkel et al., 2009). The first column is the forensic path within the evidence file; the second column is the extracted email address; the third column is the email address in context (unprintable characters are represented as underbars). These email addresses are extracted from executables found within the Linux operating system and as a result do not constitute private information or human subject data.

search engines; the keyword substring is extracted with the regular expression parenthesis operator:

search.*[?&/;fF][pq]=([ˆ&/]+) (1)

Experience has shown that a histogram of the extracted search terms dramatically improves their usefulness to the investigator, since items of import tend to be present multiple times on the subject media.

Histograms of search terms are particularly useful when conducting an investigation, as they can reveal the intent of the computer's user (§6.1). Individuals frequently engage in repeated searches for items of interest. However, the tool explicitly does not suppress low-density information, since it may be quite valuable in some cases. (§3.7 discusses approaches for weighting features.)

For example, at the 2008 murder trial of Neil Entwistle, prosecutors introduced evidence that Entwistle had performed Internet searches for murder techniques just three days before his wife and child were found dead (Associated Press). The *bulk_extractor*'s ability to identify, extract and make histograms of search terms has been used in court with some success (§6.1).

Once the search terms are extracted, *bulk_extractor* creates a histogram of the extracted terms.

Critical to using *bulk_extractor*'s reports in court is the fact that the feature file clearly identifies the physical location on the media from which the search terms were recovered; this location allows the evidence to be rapidly located and re-analyzed using other tools.

3.7. *Context-sensitive stop lists*

Many of the email addresses, phone numbers and other identifiers on a hard drive are distributed as part of the operating system or application programs. For example, previous work identified the email address mazrob@panix.com as being part of the Windows 95 Utopia Sound Scheme (Garfinkel, 2006). One way to suppress these common features is to weigh each feature by its inverse corpus frequency, a novel application of the well-known TF-IDF approach used in information retrieval (Jones, 1972).

For reasons not anticipated in Garfinkel (2006), it is not possible for many organizations to create a single list of all email addresses extracted from every processed disk. Instead, many organizations manually maintain lists of email addresses and domain names to ignore based on examiner experience. Such stop lists can also be readily produced from default installs of popular operating systems.

There are a staggering number of email addresses and URLs in some OSes. For example, Fedora Core 12 contains nearly 14 thousand distinct email addresses (Table 2). Additional email addresses and URLs are also present in application programs. Clearly, a forensic analyst who does not employ stop lists will be overwhelmed by such information. However, there is also a significant danger in naïvely employing stop lists: They can provide criminals with the ability to escape detection by using an email address associated with an operating system. Given that it is relatively easy to get an arbitrary email address embedded in open source programs, this is a significant and previously unrecognized risk when using stop lists.

One solution, introduced with *bulk_extractor*, is the *context-sensitive stop list*. The key insight is that email addresses such as mazrob@panix.com should only be ignored when they are encountered in the context of operating system files—elsewhere they should be reported. So instead of a stop list containing just the email addresses found in default operating system installs, *bulk_extractor* uses a stop list that contains the *local context*—that is, the characters that appear before and after the feature to be suppressed.

Table 1 shows the results of histogram processing on the *nps-2009-domexusers* disk image before and after the application of a context-sensitive stop list produced from several default Windows XP, 2000 and 2003 installations. The stop list removes email addresses clearly associated with certificate authorities (*e.g.* ips@mail.ips.edu and premium-server@thawte.com), but leaves those email addresses associated with the scenario.

Items on a stop list are not suppressed entirely. Instead, they are reported in specially designated feature files. This allows the examiner to manually review the stopped items to verify that no case-specific information was inadvertently excluded. It also allows validation of the stop list processing.

3.8. *Alert lists ("Go lists")*

Whereas a Stop List is a list of features that should be suppressed, an Alert List (also known has a "Go List") is a list of features that, when found, may warrant further investigation

Table 1 – Histogram analysis of the *nps-2009-domexusers* disk image before and after the application of the context-sensitive stop list. The email address 49091023.6070302@gmail.com is the Message-ID of a webmail message. This histogram was generated with an earlier version of *bulk_extractor*.

Before		After	
Freq	Email	Freq	Email
$n = 579$	domexuser1@gmail.com	$n = 579$	domexuser1@gmail.com
$n = 432$	domexuser2@gmail.com	$n = 432$	domexuser2@gmail.com
$n = 340$	domexuser3@gmail.com	$n = 340$	domexuser3@gmail.com
$n = 268$	ips@mail.ips.es	$n = 192$	domexuser2@live.com
$n = 252$	premium-server@thawte.com	$n = 153$	domexuser2@hotmail.com
$n = 244$	CPS-requests@verisign.com	$n = 146$	domexuser1@hotmail.com
$n = 242$	someone@example.com	$n = 134$	domexuser1@live.com
$n = 237$	inet@microsoft.com	$n = 91$	premium-server@thawte.com
$n = 192$	domexuser2@live.com	$n = 70$	talkback@mozilla.org
$n = 153$	domexuser2@hotmail.com	$n = 69$	hewitt@netscape.com
$n = 146$	domexuser1@hotmail.com	$n = 54$	DOMEXUSER2@GMAIL.COM
$n = 134$	domexuser1@live.com	$n = 48$	domexuser1%40gmail.com@imap.gmail.com
$n = 115$	example@passport.com	$n = 42$	domex2@rad.li
$n = 115$	myname@msn.com	$n = 39$	lord@netscape.com
$n = 110$	ca@digsigtrust.com	$n = 37$	49091023.6070302@gmail.com

and should immediately be brought to the attention of an investigator. The *bulk_extractor* has provisions for a global alert list. Such a list could be email addresses known to be associated with fraud, IP addresses known to be associated with malware, and so on.

4. Implementation

The *bulk_extractor* is a command-line program that implements the design described in the previous section and has been deployed to several production environments around the world. Version 1.2.0 of *bulk_extractor* consisted of 12,255 lines of C++ code and 1086 lines of GNU flex code. Fig. 1 depicts a conceptual overview of the program.

4.1. *Functional modules*

The current implementation of *bulk_extractor* consists of five modules:

1. The **initialization module** verifies command line parameters, and creates the analysis thread pool.
2. The **image processing module** reads the disk image, extracts a series of pages, and passes each page off to a thread in the thread pool.
3. The **analysis thread pool** operates over multiple threads, each of which receives an incoming page and processes it with one or more feature scanners.
4. The **feature scanners** process each buffer and identifies features that can be recovered.
5. The **feature recording module** records features identified by the scanners in one or more feature files.

4.2. *Scanner implementation*

Purpose-built scanners using hand-tuned rules extract forensic information, an approach validated in the 1990s by natural language researchers (Nadeau and Sekine, 2007). The email and accounts scanners are both implemented as

Table 2 – Number of unique features of each type found by *bulk_extractor* on various base operating system installs. All of the hits in the CCNs column appear to be false positives. These numbers were generated with an earlier version of *bulk_extractor*.

VM	CCNs	Domains	Email	Exif	RFC822	Tel.	URLs	ZIPs
fedora12-64	1	13,973	21,612	119	2017	662	75,555	55,172
macos10.6	35	2432	2669	909	485	256	6781	55,793
redhat54-ent-64	0	12,345	17,669	36	2052	3773	25,078	20,749
win2003-32bit	0	475	227	6	41	65	7878	149
win2003-64bit	0	330	172	5	40	42	7421	163
win2008-r2-64	64	565	254	37	105	77	8196	91
win7-enterprise-32	68	699	365	149	110	77	6800	91
win7-ultimate-64	68	677	371	145	100	78	6606	105
winXP-32bit-sp3	0	492	306	7	61	132	8916	296
winXP-64bit	0	404	262	17	68	54	7869	296

a series of regular expressions compiled using GNU Flex (Fig. 3). This technique creates object code that can be megabytes in size, but nevertheless runs quickly. Another scanner implements the AES key-finding algorithm developed by Halderman et al. (2009).

Early scanners produced unacceptably high levels of false positives when processing PDFs, TIFFs, and other file types containing long runs of formatted numbers. The false positives were decreased with additional hand-tuned filters that examine the local context in which the features are found. For example, EnCase 6.2.1 identifies the string "`Box [−568 −307 2000 1006] /FontName`" as containing the US phone number "`307 2000`", but *bulk_extractor* does not, as it recognizes that the numbers are part of a larger group that does not conform to the US phone number pattern.

The *bulk_extractor* uses a straightforward API with a single callable function to initialize each scanner, process bulk data, and perform post-processing. Scanners can be linked into the *bulk_extractor* executable or loaded at run time. This makes it easy for organizations to extend *bulk_extractor* using proprietary technology while still maintaining an open source base that is usable by the general community.

The current scanners distributed with the open source *bulk_extractor* are shown in Table 3.

4.3. *The margin*

While it is straightforward to process a disk image block-by-block or page-by-page, occasionally important features cross block or page boundaries. Several file carvers that process bulk data simply discard features that cross these boundaries. Analysts interviewed were generally unaware that their tools might systematically discard boundary-crossing features: upon learning of this behavior, they pronounced it unacceptable.

The *bulk_extractor* avoids this problem by appending to each buffer a *margin* of additional bytes from the following page of the evidence file. The implementation thus maintains two lengths for each buffer: the *page size* and the *margin size*. These values may be specified by the user; the defaults are 16 MiB and 1 MiB respectively. These defaults were determined experimentally but can be readily changed as necessary.

The design of the margin poses minimal performance overhead: while features can extend into the margin, once the analysis point passes into the margin, the scanner stops. Thus, features that begin in the margin are never considered. The only significant overhead is the double-reading of the data in the margin itself, although this data is typically cached, minimizing the performance impact.

The margin needs to be large enough so that any feature or compressed region that begins near the end of the current page will likely fit within the margin. The value of 1 MiB is sufficient to cover most compressed streams. However, in cases involving source code analysis, where there are likely to be large *.tar.gz* archives, it is necessary to use a margin that is larger than the largest anticipated archive.

4.4. *Parallelizing* bulk_extractor

Many authors have noted that it is vital for forensic tools to adopt parallelism both in order to keep up with increasing forensic collection sizes and to make the most efficient use of modern hardware (*e.g.* Ayers, 2009; Richard and Roussev, 2006). Vendors have been slow to parallelize their tools because of the complexity of managing multiple analyzers and combining the results in a thread-safe manner. The approach of treating the disk image as a series of independent pages turns feature

```
END      ([^0-9e\.]|(\.[^0-9]))
BLOCK    [0-9]{4}
DELIM    ([- ])
SDB      ([45][0-9][0-9][0-9]{DELIM})
DB       ({BLOCK}{DELIM})
%%
[^0-9]{SDB}{DB}{DB}{DB}{BLOCK}/{END} {
    /* #### #### #### #### --- most credit card numbers */
    /* don't include the non-numeric character in the hand-off */
    if(validate_ccn(yytext+1,yyleng-1)){
        ccn_recorder->write_buf(pos0,buf,bufsize,pos+1,yyleng-1);
    }
    pos += yyleng;
}

fedex[^a-z]+[0-9][0-9][0-9][0-9][- ]?[0-9][0-9][0-9][0-9][- ]?[0-9]/{END} {
    ccn_recorder->write_buf(pos0,buf,bufsize,pos,yyleng);
    pos += yyleng;
}
```

Fig. 3 – These excerpts from *bulk_extractor*'s *scan_accts.flex* input file for GNU flex (The Flex Project, 2008) shows how multiple regular expressions are combined with external validators to extract credit card numbers, FedEx account numbers, and other types of information. Although these regular expressions must be manually created, tuned and maintained, they offer high speed and an astonishingly low false positive rate.

Table 3 – Scanners included with *bulk_extractor* 1.2.

Name	Recognizes
Basic Scanners:	
scan_accts	Credit card numbers, phone numbers, and other formatted numbers
scan_aes	AES key schedules in memory
scan_email	RFC822 headers, HTTP Cookies, hostnames, IP addresses, email addresses, URLs
scan_exif	JPEG EXIF headers
scan_find	User-provided regular expression searches
scan_gps	Garmin-formatted XML containing GPS coordinates
scan_json	Javascript Object Notation (JSON)
scan_kml	KML file recovery
scan_net	IP and TCP packets in virtual memory; creates libpcap files
scan_winprefetch	Windows prefetch files
scan_wordlist	Words (for password cracking)
Recursive Scanners:	
scan_base64	BASE64 coding
scan_gzip	gzip (Deutsch, 1996a) compressed files (including HTTP streams)
scan_hiberfile	Windows hibernation file decompression
scan_pdf	DEFLATE (Deutsch, 1996b) compressed streams in PDF files and text extraction
scan_zip	Components of ZIP compressed files

extraction into an “embarrassingly parallel” problem, as each page can be processed independently once features that cross page boundaries are properly handled (§4.3).

A series of tests using the 2.1 GB *nps-2009-ubnist1* disk image (Garfinkel et al., 2009) validates *bulk_extractor*'s multi-threading approach and demonstrates the relative performance of multiple threads under Windows and MacOS. The test machine was a MacBook Pro with 8GiB of RAM and 2 Ghz four-core Intel i7 processor, and an SSD; the machine was reboot between each trial to assure that the disk image was not cached in the system's RAM.

With a single thread, MacOS required on average 3998 s to analyze the disk image, for an average performance of 0.53 MB/sec. With four analyzing threads the performance increased to 2.5 MB/sec—roughly a linear speedup, as evidenced by the first part of Fig. 4. Cores 5 through 8 are hyperthreaded, meaning that the processor runs instructions on these virtual “cores” when the various functional units are not otherwise in use. Not surprisingly, there is no longer a strict linear speedup, although the region of the graph as the number of cores are increased from 5 to 8 is itself linear. With eight analyzing threads performance increased to 3.0 MB/s. Thus, the additional four hyperthreaded “cores” provide roughly the power as a single physical core. This is uncharacteristically poor performance for hyperthreading, and it argues that the *bulk_extractor* threads are making excellent use of the CPU's functional units, leaving few resources available for the virtualized cores. Additional threads resulted in lower performance, presumably due to contention. For this reason, *bulk_extractor* automatically determines the correct number of analyzing threads to use when it runs, freeing the examiner from this task.

Although linear speedup is also observed under Windows, the peak performance was roughly two-thirds of the same hardware running MacOS. No speedup is observed when the virtual cores are added. There is no good explanation for the poor performance of *bulk_extractor* under the Windows operating system, although other programs that run under both Windows and MacOS behave similarly.

A realistic comparison with EnCase was performed using the 40 GB *nps-2009-domexusers* disk image as test data and a typical examiner's machine: a dual-processor Xenon X5650 at 2.67 Ghz (12 physical cores, 12 hyperthreaded cores), with 12 GiB RAM running Windows 7 Professional. The test was simple: extract and report all of the email addresses in the disk image. The results (Fig. 5) indicate that the multi-threaded *bulk_extractor* extracted email addresses from the forensic test disk image 10× faster than EnCase 6.2.1 running on the same hardware, and was only 5% slower than simply running *ewfexport*, *strings* and *grep* (Fig. 5) while performing significantly more analysis. As will be shown in §5.1,

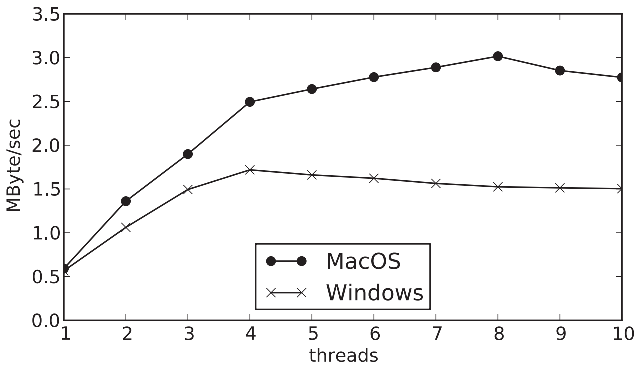

Fig. 4 – Speed of *bulk_extractor* to process the 2.1 GB disk image *nps-2009-ubnist1.gen3* as a function of number of threads and threading model. Reference computer was a MacPro with a 2 GHz Intel Core i7 CPU with 8 GB of 1333 MHz DDR3 RAM. The i7 has four physical cores and four virtual cores with hyperthreading. Notice that the thread pool shows linear performance improvement as the utilization of physical cores increases, and then again linear improvement (at a slower rate) as utilization of the hyperthreading virtual cores increases. Increasing the number of threads beyond the number of virtual cores results in no further improvement of performance.

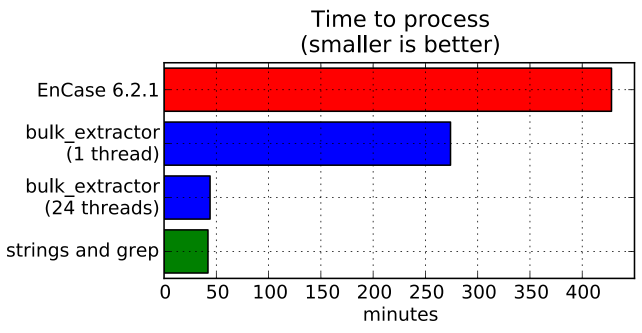

Fig. 5 – Guidance Software's EnCase 6.2.1 takes 7 h, 8 min to extract email addresses and other information from the 42 GB disk image *nps-2009-domexusers*. The *bulk_extractor* performs the same task in 274 min in single-threaded mode, and in 44 min in multi-threaded mode. The combination of *ewfexport*, *strings* and *grep* runs fastest, in just 42 min, but unlike *bulk_extractor* cannot extract email addresses from compressed data regions and fails to extract other information such as URLs and credit card numbers. Tests were performed on a MacPro with 16 GiB 1066 MHz DDR3 and two 2.26 GHz quad-core Intel Xeon processors, for 12 physical cores and 12 virtual cores with hyperthreading. The *strings* and *grep* commands were run under the Cygwin POSIX emulation environment. This graph was generated with a previous version of *bulk_extractor*.

bulk_extractor also found significantly more email addresses than either EnCase or *strings*, because of its ability to extract strings from compressed regions and from fragments of PDF and hibernation files. Furthermore, *bulk_extractor* extracts binary information that neither EnCase nor *strings* currently finds, such as AES encryption keys.

Both EnCase and *bulk_extractor* produce files that can be used to dramatically speed subsequent analysis. EnCase produces an index file that can be used for full-text searching, whereas *bulk_extractor* produces feature files that can be rapidly searched using the Java-based *bulk_extractor* user interface. Thus, both tools allow for searched email addresses to be used as keys for finding additional information such as the time a message was sent or the body of an email.

5. Validation

This section discusses how *bulk_extractor* has been tested with constructed test data, clean operating system installs, and real data.

5.1. *Constructed test drives*

Although the recovery of email addresses from disk drives is a common forensic task, there are no standard tests or data sets.

Faced with this lack of test images, a series of test documents was created using various office productivity applications. Each document contained a single descriptive email address. Some of the documents were placed in a cleared FAT32 disk image; others were used to generate PDF files that were placed on the disk image. The disk image was then subjected to analysis with *bulk_extractor*, EnCase 6.2.1, and finally with the traditional combination of *strings* followed by *grep* as a control.

For every target, *bulk_extractor* finds more email addresses than either EnCase or the control, primarily because of its ability to detect and recursively re-analyze compressed data (Table 4). The only email addresses that *bulk_extractor* fails to find are those in PDF files generated by Microsoft Office. Analysis reveals that while Apple's TextEdit preserves many strings (Fig. 6) in the PDF file, Microsoft Word does not (Fig. 7). Carrier (2010) has stated that the only reliable way to extract text from PDFs is to render the page and process it with an optical character recognition program, a strategy that *bulk_extractor* does not employ.

5.2. *Base OS installs*

It is important to test forensic tools with base installations of operating systems: characterizing the tool's behavior on base installs helps to predict how the tool will behave on the same OS files when they are present on subject media.

Analyzing Linux systems with *bulk_extractor* yields thousands of domains, email addresses, URLs, telephone numbers, and components of ZIP files. Inspection reveals that the domains and email addresses were largely those of Linux developers; the URLs were typically those of open source software update distribution points, web-based services used for software updates and XML schema; the telephone numbers are from software license agreements. For example, the most common phone number (found 47 times) in the Fedora 12 release was (412) 268-4387 (the number of the Carnegie Mellon University office of Technology Transfer, present in several copyright licenses). The most common false positives come from arrays of numbers in PDF files that are grouped in patterns resembling phone numbers. Adding a rule to discard any phone number preceded by a pair of 3 or 4 digit numbers or followed by the characters `_/Subtype` largely solved this problem.

The Windows operating systems are, comparatively speaking, quite clean. In the default Windows installations there are only a few email addresses, but there are a significant number of strings that appear to be "IP addresses" which are actually version numbers and SNMP OIDs. The accuracy rate of the IP address scanner could be improved by considering a larger context.

The results from running *bulk_extractor* on base OS installs can be used for creating context-sensitive stop lists (§3.7).

5.3. *Prevalence of compressed email addresses*

To gauge the value of *bulk_extractor*'s optimistic decompression, used disks and USB storage devices still containing data from their original users were analyzed for the presence of email addresses that could be recovered using gzip or ZIP decompression and that were not otherwise present.

The disk images, taken from the Real Data Corpus (Garfinkel et al., 2009), come mostly from China, India, Israel and Mexico. Previous research has established that used

Table 4 – Email addresses were stored in sample documents using specific applications. The documents were copied into a disk image. The resulting image was analyzed separately with *strings* and *grep*, EnCase, and *bulk_extractor*. This table shows which email addresses could be recovered.

email address	Application (encoding)	*strings* & *grep*	EnCase	BE
plain_text@textedit.com	Apple TextEdit (UTF-8)	✓	✓	✓
plain_text_pdf@textedit.com	Apple TextEdit print-to-PDF (/FlateDecode)			✓
rtf_text@textedit.com	Apple TextEdit (RTF)	✓	✓	✓
rtf_text_pdf@textedit.com	Apple TextEdit print-to-PDF (/FlateDecode)			✓
plain_utf16@textedit.com	Apple TextEdit (UTF-16)		✓	✓
plain_utf16_pdf@textedit.com	Apple TextEdit print-to-PDF (/FlateDecode)			✓
pages@iwork09.com	Apple Pages '09	✓	✓	✓
pages_comment@iwork09.com	Apple Pages (comment) '09			✓
keynote@iwork09.com	Apple Keynote '09			✓
keynote_comment@iwork09.com	Apple Keynote '09 (comment)			✓
numbers@iwork09.com	Apple Numbers '09			✓
numbers_comment@iwork09.com	Apple Numbers '09 (comment)			✓
user_doc@microsoftword.com	Microsoft Word 2008 (Mac) (.doc file)	✓	✓	✓
user_doc_pdf@microsoftword.com	Microsoft Word 2008 (Mac) print-to-PDF			
user_docx@microsoftword.com	Microsoft Word 2008 (Mac) (.docx file)			✓
user_docx_pdf@microsoftword.com	Microsoft Word 2008 (Mac) print-to-PDF (.docx file)			
xls_cell@microsoft_excel.com	Microsoft Word 2008 (Mac)	✓	✓	✓
xls_comment@microsoft_excel.com	Microsoft Word 2008 (Mac)			✓
xlsx_cell@microsoft_excel.com	Microsoft Word 2008 (Mac)			✓
xlsx_cell_comment@microsoft_excel.com	Microsoft Word 2008 (Mac) (Comment)			✓
doc_within_doc@document.com	Microsoft Word 2007 (OLE .doc file within .doc)	✓	✓	✓
docx_within_docx@document.com	Microsoft Word 2007 (OLE .doc file within .doc)	✓	✓	✓
ppt_within_doc@document.com	Microsoft PowerPoint and Word 2007 (OLE .ppt file within .doc)	✓	✓	✓
pptx_within_docx@document.com	Microsoft PowerPoint and Word 2007 (OLE .pptx file within .docx)			✓
xls_within_doc@document.com	Microsoft Excel and Word 2007 (OLE .xls file within .doc)	✓	✓	✓
xlsx_within_docx@document.com	Microsoft Excel and Word 2007 (OLE .xlsx file within .docx)			✓
email_in_zip@zipfile1.com	text file within ZIP			✓
email_in_zip_zip@zipfile2.com	ZIP'ed text file, ZIP'ed			✓
email_in_gzip@gzipfile.com	text file within gzip			✓
email_in_gzip_gzip@gzipfile.com	gzip'ed text file, gzip'ed			✓

computer systems, purchased at second-hand computer stores, in open-air markets, and other venues, frequently contain information left from their former users. As a result, the data from used computer systems can be productively used as a surrogate for digital media acquired during the course of law enforcement operations. Information collected on the secondary market in this fashion has the advantage of being usable in research, as it does not contain law-enforcement sensitive information, provided that sufficient privacy controls are in place. This study further confirms those research findings.

```
q Q q 72 300 460 420 re W n /Gs1 gs /Cs1 c
s 1 sc 72 300 460 420 re f 0 sc./Gs2 gs q
1 0 0 -1 72 720 cm BT 10 0 0 -10 5 10 Tm /
F1.0 1 Tf (plain_text_pdf@textedit.com).Tj
 ET Q Q
```

Fig. 6 – An inflated stream from a PDF file created using Apple's TextEdit application. The original document contained the string "plain_text_pdf@textedit.com". Notice that the email address is preserved in the output.

To gauge the age of each drive, the *Date:* headers in email messages found on the disks were also extracted with *bulk_extractor*. The date of each disks' last activity was computed by averaging the five most recent timestamps. Visual inspection of a randomly chosen sample confirmed that the extracted timestamps did in fact correspond to actual times.

Email addresses present on more than a single drive were discarded. The remaining addresses were tabulated according to whether each email address appeared in plaintext or in a compressed stream. There were 865 drives that contained email addresses, of which 431 also contained timestamps. Table 5 presents, for each year, the number of drives recovered for that year, the total number of email addresses, the number of email addresses encoded with each compression algorithm, and the number of email addresses only present on a single drive.

As shown, there are a significant number of email addresses that can *only* be recovered by optimistically decompressing forensic data, demonstrating once again that the recall performance of forensic tools can be significantly improved through the use of optimistic decompression.

```
q Q q 12 12.00002 588 768 re W n /Cs1 cs 0
 0 0 sc q 0.24 0 0 0.24 90 708.96.cm BT 50
 0 0 50 0 0 Tm /F1.0 1 Tf (This is a test
 ) Tj ET Q q 0.24 0 0 0.24 156.3352 708.96
.cm BT 50 0 0 50 0 0 Tm /F1.0 1 Tf (---) T
j ET Q q 0.24 0 0 0.24 168.3227 708.96.cm
BT 50 0 0 50 0 0 Tm /F1.0 1 Tf ( ) Tj ET Q
 0 0 1 sc q 0.24 0 0 0.24 171.3227 708.96.
cm BT 50 0 0 50 0 0 Tm /F1.0 1 Tf (user_do
c) Tj ET Q q 0.24 0 0 0.24 214.6423 708.96
.cm BT 50 0 0 50 0 0 Tm /F1.0 1 Tf (_pdf)
Tj ET Q q 0.24 0 0 0.24 236.6382 708.96.cm
 BT 50 0 0 50 0 0 Tm /F1.0 1 Tf (@microsof
tword.com) Tj ET Q 171.36 707.28 170.64 0.
4799805.re f 0 0 0 sc q 0.24 0 0 0.24 342.
0088 708.96 cm BT 50 0 0 50 0 0 Tm /F1.0.1
 Tf [ (   Really) -1 (.) ] TJ ET Q q 0.24
0 0 0.24 385.3523 708.96 cm BT.50 0 0 50 0
 0 Tm /F1.0 1 Tf ( ) Tj ET Q q 0.24 0 0 0.
24 90 695.28 cm BT 50 0 0 50 0 0.Tm /F1.0
1 Tf ( ) Tj ET Q Q
```

Fig. 7 – An inflated stream from a PDF file created using Microsoft Word 2008 for Macintosh. The original document contained the string "This is a test—user_doc_pdf@microsoftword.com Really." Notice that the email address is split into three pieces.

6. Case studies

This section discuss two cases in which *bulk_extractor* provided timely information that proved critical to real-world investigations.

6.1. *Credit card fraud*

In the spring of 2010 the San Luis Obispo, CA, County District Attorney's Office "filed charges of credit card fraud case and possession of materials to make fraudulent credit cards against two individuals" (Lehr, 2010). The day before the suspects' preliminary hearing, an evidence technician at the SLO police department was given a 250 GB hard drive that had been seized in conjunction with the suspects' arrest. The technician was told to find evidence that tied the suspects to the alleged crime; the defense was expected to claim that the computer belonged to an unindicted associate, and that the defendants lacked computer skills.

An early version of *bulk_extractor* was able to analyze the hard drive in roughly 2.5 h. The program quickly established that:

- More than 10,000 credit card numbers were present on the hard drive.
- The most common email address clearly belonged to the primary defendant, disproving his contention that he had no connection to the drive and helping to establish the defendant's possession of the credit card numbers.
- The most common Internet search queries concerned credit card fraud and bank identification numbers. This helped to establish the defendant's intent to commit fraud (Lehr, 2010).

Based on the reports generated by *bulk_extractor* and the testimony of the technician, the court concluded that the District Attorney had met the burden of proof to hold the suspects pending trial.

It is unlikely that such high quality reporting could have been generated so quickly (and with so little effort on the part of the investigator) with a conventional forensic tool. Given

Table 5 – A study of 431disk drives acquired around the world shows that there is a significant presence of email addresses that can only be recovered by decompressing ZIP and gzip-compressed data streams. The email addresses in this table are restricted to those that were found on a single hard drive in the study.

		Email addresses found				
Year	Drives	Total	In ZIP	Uniquely in a single ZIP stream	In gzip	Uniquely in a single gzip stream
1991	1	57	0	0	0	0
1992	1	2663	0	0	16	0
1993	2	33,352	0	0	255	20
1994	4	18,401	310	0	1096	421
1995	13	76,469	53	14	1353	71
1996	19	314,340	10	1	43,567	1532
1997	16	755,597	276	86	31,057	1066
1998	30	252,811	66	4	297	14
1999	30	1,167,208	79	4	118	11
2000	51	2,709,526	67,221	306	199,063	3615
2001	51	1,180,587	360	109	9118	710
2002	46	6,336,891	414	57	341,810	9178
2003	44	1,654,880	429	95	19,444	563
2004	49	2,746,356	1088	222	30,705	9989
2005	27	351,238	75	3	2656	127
2006	26	326,480	56	7	1370	40
2007	17	828,229	301	0	3319	388
2008	4	799,127	9	2	2171	59

the time pressures, it is quite possible that the suspects would have been granted bail without the *bulk_extractor* results and might have left the country, complicating prosecution.

6.2. *ATM fraud*

Also in 2010, A 250 GB disk drive was recovered from individuals suspected of placing credit-card "skimmers" and pinhole cameras at ATM machines in a major US city. Police needed to rapidly supply the banks with a list of the compromised credit card numbers so that the accounts could be closed.

The *bulk_extractor* completed its processing after just 2 h on a quad-core computer. The banks in question were provided with the *ccns.txt* output file (a list of credit-card numbers found on the drive), and were able to immediately shut down the accounts before fraud could be committed.

As in the previous case, it is unlikely that conventional file-based tools could have found the compromised credit card numbers in so short a time. Although a program such as *strings* might have extracted some of the email addresses and credit card numbers, it would not have created the histograms or suppressed false-positives. (The *bulk_extractor* credit card scanner includes substantial code for suppressing numbers that satisfy the Luhn (1960) algorithm but are nevertheless not valid. Because the Lhun algorithm is a single digit checksum, 1-out-of-10 sequences of random digits that are the correct length satisfy the check. For example, 5555-4444-3333-2226 satisfies the Lhun algorithm. The *bulk_extractor* suppresses the number because of the abnormal distribution of digits. Additional checks that examine the local context are described in Section 4.2.) Being able to find the card numbers in short order was vital to blocking the cards before fraud could be committed.

7. Limitations and future work

Bulk data analysis and the *bulk_extractor* are designed to complement traditional forensic approaches, not replace them. This section discusses specific limitations that have been encountered and the outlook for this technology.

7.1. *Theoretical limitations of bulk data analysis*

The primary limitation of bulk data analysis is that compressed objects fragmented across multiple locations are difficult to recover. However, with the exception of log files, most forensically interesting files are not fragmented (Garfinkel, 2007). While most log files are fragmented, most are stored without compression, allowing the various fragments to be matched and recombined. In the rare cases that compressed objects are fragmented, Memon has shown that decompressing the compressed streams can be used as a tool for fragment reassembly (Sencar and Memon, 2009).

7.2. *Unicode and internationalization*

Unicode and non-Latin character sets pose challenges for bulk data processing and extraction. An added complication is that ASCII and Unicode are not the only types of localized strings likely to be found. Data may be coded using a Windows Code Page, Big5, EUC-JP, GB18030, Shift-JIS, or other coding schemes.

Fortunately, most email addresses and URLs encountered today are in simple ASCII or ASCII encoded as UTF-16, both of which are handled by *bulk_extractor*. In the near term, *bulk_extractor* will be modified to incorporate other strategies.

7.3. *Characterization of bulk data*

In addition to feature extraction, it is also useful at times to report on the overall characteristics of bulk data. For this reason a future version of *bulk_extractor* will implement an algorithm, previously described, for distinguishing compressed data from encrypted data (Garfinkel et al., 2010), and will report the amount of encrypted data present on the digital media being examined.

7.4. *Correlation with other kinds of forensic data*

Much of the information that is produced by a run of *bulk_extractor* can be correlated with other kinds of forensic information. For example, the output of *bulk_extractor* can be readily used for *cross-drive analysis* (Garfinkel, 2006). It is also possible to use network contextual data to provide linkage to other kinds of network forensics data. Such linkage can be critical in scene reconstruction for the eventual courtroom presentation.

8. Conclusion

This paper discusses a number of advances in the field of bulk data analysis and presents the design and implementation of the *bulk_extractor*, a forensic tool that extracts forensic features from bulk data. Information extracted by *bulk_extractor* is reported in *feature files* that indicate where each feature was found in the source file. The *bulk_extractor* also performs histogram analysis. Histograms can be used to rapidly determine the most common email addresses and most frequent search terms present on a hard drive. The tool optimistically decompresses compressed data that it encounters.

The *bulk_extractor* was developed as a research platform but has found use in actual cases. The program's source code, ancillary programs, and pre-compiled executables for Windows are available for download at http://forensicswiki.org/wiki/bulk_extractor/. The code is public domain and may be freely incorporated into other open source or commercial applications.

The constructed drive image can be downloaded from http://digitalcorpora.org.

Acknowledgments

The author gratefully acknowledges the participation of John Lehr at the San Luis Obispo Police Department for his use of early *bulk_extractor* prototypes and his permission to print here the stories regarding the use of *bulk_extractor* in actual cases. Rob Beverly, Beth Rosenberg and others reviewed earlier versions of this paper and provided useful feedback. Anonymous reviewers provided additional information and criticism that was quite thought-provoking and useful in refining this article. This work was supported by NSF Award DUE-0919593 and the Department of Defense. The views expressed in this document are those of the author and do not represent those of the Department of Defense or the US Government.

REFERENCES

AccessData. Forensic toolkit (FTK), http://accessdata.com/products/computer-forensics/ftk; 2011 [accessed 03.12.11].

AccessData Corporation. AccessData supplemental appendix: regular expression searching; 2008.

Aera Network Security. Decompression bomb vulnerabilities, http://www.aerasec.de/security/advisories/decompression-bomb-vulnerability.html; 2009 [accessed 03.12.11].

Associated Press. Testimony expected on Neil Entwistle's computer activity the day of the murders of his wife, baby. Associated Press. http://www.foxnews.com/story/0,2933,368604,00.html [accessed 03.12.11].

Ayers D. A second generation computer forensic analysis system. In: Proceedings of the 2009 Digital Forensics Research Workshop, DFRWS; 2009.

Beverly R, Garfinkel S, Cardwell G. Forensic carving of network packets and associated data structures. In: Proceedings of the eleventh annual DFRWS conference, vol. 8. Elsevier; 2011.

Bird RM, Tu JC, Worthy RM. Associative/parallel processors for searching very large textual data bases. SIGMOD Record 1977; 9:8–9, http://doi.acm.org/10.1145/965645.810247 [accessed 03.12.11].

Bunting S. EnCE: the official EnCase certified examiner study guide. Sybex; 2008.

Committee on Identifying the Needs of the Forensic Science Community. Strengthening forensic science in the United States: a path forward. National Research Council; 2009.

Carrier B. Extracting searchable text from arabic pdfs, http://www.basistech.com/knowledge-center/forensics/extracting-text-from-arabic-pdf.pdf; 2010 [accessed 03.12.11].

CFTT Program. Forensic string searching tool requirements specification. National Institute of Standards and Technology (NIST), http://www.cftt.nist.gov/ss-req-sc-draft-v1_0.pdf; Jan. 24, 2008 [accessed 03.12.11].

CFTT Program. Overview dfr test images. National Institute of Standards and Technology, http://www.cfreds.nist.gov/dfr-testimages.html; 2012 [accessed 09.04.12].

Cohen M. PyFlag: an advanced network forensic framework. In: Proceedings of the 2008 Digital Forensics Research Workshop, DFRWS; 2008.

Collange S, Daumas M, Dandass YS, Defour D. Using graphics processors for parallelizing hash-based data carving. In: Proceedings of the 42nd Hawaii international conference on system sciences, http://hal.archives-ouvertes.fr/docs/00/35/09/62/PDF/ColDanDauDef09.pdf; 2009 [accessed 03.12.11].

Corey V, Peterman C, Shearin S, Greenberg MS, Van Bokkelen J. Network forensics analysis. IEEE Internet Computing 2002;6(6): 60–6.

Cornell University IT Security Office. Spier 2.9.5 for windows, http://www2.cit.cornell.edu/security/tools/; 2008 [accessed 03.12.11].

Daubert v. Merrell Dow Pharmaceuticals, 509 US 579; 1993.

Deutsch LP. RFC 1952: GZIP file format specification version 4.3. status: INFORMATIONAL; 1996a.

Deutsch LP. RFC 1951: DEFLATE compressed data format specification version 1.3. status: INFORMATIONAL; 1996b.

Garfinkel SL. Forensic feature extraction and cross-drive analysis. In: Proceedings of the 6th annual Digital Forensic Research Workshop (DFRWS). Lafayette, Indiana: Elsevier, http://www.dfrws.org/2006/proceedings/10-Garfinkel.pdf; 2006 [accessed 03.12.11].

Garfinkel SL. Carving contiguous and fragmented files with fast object validation. In: Proceedings of the 7th annual Digital Forensic Research Workshop (DFRWS), vol. 4. Pittsburgh, PA: Elsevier; 2007. p. 2–12.

Garfinkel SL. Automating disk forensic processing with SleuthKit, XML and Python. In: Proceedings of the fourth international IEEE Workshop on Systematic Approaches to Digital Forensic Engineering, IEEE. Oakland, CA: IEEE; 2009. p. 73–84.

Garfinkel S. Digital forensics: the next 10 years. In: Proceedings of the tenth annual DFRWS conference, vol. 7; 2010. Portland, OR.

Garfinkel S, Migletz J. New XML-based files: implications for forensics. IEEE Security & Privacy Magazine 2009;7(2): 38–44.

Garfinkel SL, Malan DJ, Dubec K-A, 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.

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

Guidance Software, Inc.. EnCase forensic, http://www.guidancesoftware.com/forensic.htm; 2011 [accessed 03.12.11].

Guo Y, Slay J, Beckett J. Validation and verification of computer forensic software tools—searching function. Digital Investigation 2010;6:S12–22.

Halderman JA, Schoen SD, Heninger N, Clarkson W, Paul W, Calandrino JA, et al. Lest we remember: cold-boot attacks on encryption keys. Communications of the ACM 2009;\52: 91–8, http://doi.acm.org/10.1145/1506409.1506429 [accessed 03.12.11].

Jones KS. A statistical interpretation of term specificity and its application in retrieval. Journal of Documentation 1972;28: 11–21, http://www.soi.city.ac.uk/ ~ser/idfpapers/ksj_orig.pdf [accessed 03.12.11].

Konstantopoulos C, Mamalis B, Pantziou G, Gavalas D. Efficient parallel text retrieval techniques on bulk synchronous parallel (bsp)/coarse grained multicomputers (cgm). Journal of Supercomputing 2009;48:286–318, http://portal.acm.org/citation.cfm?id=1555972.1555999 [accessed 03.12.11].

Kornblum J. First beta of md5deep version 4, http://jessekornblum.livejournal.com/276461.html; Nov. 7, 2011 [accessed 03.12.11].

Lehr J. Personal communication; Jun. 3, 2010.

Luhn HP. US patent 2,950,048, computer for verifying numbers; Aug. 23, 1960.

Lyle J. If error rate is such a simple concept, why don't I have one for my forensic tool yet?. In: Proceedings of the 10th annual DFRWS conference; 2010.

Metz J. Libewf: project info, http://sourceforge.net/projects/libewf/; 2008 [accessed 03.12.11].

Mora R-J. Digital forensic sampling, http://blogs.sans.org/computer-forensics/2010/03/29/digital-forensic-sampling/; Mar. 29, 2010 [accessed 03.12.11].

Nadeau D, Sekine S. A survey of named entity recognition and classification, http://nlp.cs.nyu.edu/sekine/papers/li07.pdf; 2007 [accessed 03.12.11].

Parsonage H. Computer forensics case assessment and triage; 2009.

Pearson S, Watson R. Digital triage forensics: processing the digital crime scene. Syngress; 2010.

Pierzchala S. Performance improvement from compression, http://newestindustry.org/2006/10/03/performance-improvement-from-compression-2/; 2006 [accessed 03.12.11].

Port80 Software. Port80's 2010 http compression survey on the top 1000 corporations' web sites, http://www.port80software.com/surveys/top1000compression/; 2010 [accessed 03.12.11].

RCF Laboratory. Annual report for fiscal year 2008; 2008.

Richard III GG, Roussev V. Next-generation digital forensics. Communications of the ACM 2006;49(2):76–80.

Rogers MK, Goldman J, Mislan R, Wedge T. Computer forensics field triage process model. In: Conference on digital forensics, security and law; 2006.

Secretary of State for the Home Department. Counter-terrorism policy and human rights: 28 days, intercept and post-charge questioning (the government reply to the nineteenth report from the joint committee on human rights session 2006–07 hl paper 157, hc 394), http://www.official-documents.gov.uk/document/cm72/7215/7215.pdf; Sep. 2007 [accessed 03.12.11].

Sencar H, Memon N. Identification and recovery of jpeg files with missing fragments. In: Digital investigation, vol. 6; 2009.

Srinivasan R. Speed web delivery with http compression, http://www.ibm.com/developerworks/web/library/wa-httpcomp/; 2003 [accessed 03.12.11].

Suiche M. Windows hibernation file for fun 'n' profit. In: Black hat 2008; 2008 [accessed 03.12.11].

The Flex Project. flex: the fast lexical analyzer, http://flex.sourceforge.net/; 2008 [accessed 03.12.11].

University of Michigan Information Technology Security Services. Tools for discovering credit card and social security numbers in computer file systems, http://safecomputing.umich.edu/tools/download/ccn-ssn_discovery_tools.pdf; 2008 [accessed 03.12.11].

Urias V, Hash C, Liebrock LM. Consideration of issues for parallel digital forensics of raid systems. Journal of Digital Forensic Practice 2008;2:196–208.

Walls RJ, Learned-Miller E, Levine BN. Forensic triage for mobile phones with dec0de. In: Proceedings of the 20th USENIX security symposium, USENIX, http://www.cs.umass.edu/elm/papers/walls11.pdf; 2011.

Simson L. Garfinkel is an Associate Professor at the Naval Postgraduate School. Based in Arlington VA, Garfinkel's research interests include computer forensics, the emerging field of usability and security, personal information management, privacy, information policy and terrorism. He holds six US patents for his computer-related research and has published dozens of journal and conference papers in security and computer forensics.