Digital media triage with bulk data analysis and *bulk_extractor*

Simson L. Garfinkel[1]

Naval Postgraduate School, 900N Glebe Arlington, VA 22207

keywords: digital forensics; bulk data analysis; bulk_extractor; opportunistic decompression; windows hibernation files; EnCase; forensic path; margin; parallelized forensic analysis

1. Introduction

With each passing year, digital forensics investigations become 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 multi-processing[7, 41], statistical sampling[**?** 36], and even legislative solutions such as extending the amount of time a suspect can be without being charged so that evidence may be analyzed[43]. But no matter what approach is used to improve performance, so long as a backlog exists, some process must be allocate limited forensic resources.

Triage is a term widely used to denote the prioritization of work according to a quality inherent in the objects being acted upon. Today many organizations prioritize their work load solely 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 prioritize resources based on the nature of a case: data collected in terrorism cases are given the highest priority so that the lab may "prevent, neutralize, dismantle, and protect against terrorist acts against Americans and assets."[32] Many local police departments use trial dates and the statute of limitations to prioritize their workload—implying that media is typically analyzed months or even years after it has been seized.

We propose expanding triage to include the results of a rapid and largely automated analysis of the media, performed when the media is first encountered. With the results of the triage it is possible to determine if the media is likely to have in-

Email address: slgarfin@nps.edu, +1 202-649-0029 (Simson L. Garfinkel)

formation of intelligence value and, therefore, should be prioritized for immediate analysis.

This article describes a number of advances we have made in *bulk data analysis*, an approach for performing digital forensics analysis that eschews file extraction, and instead focuses on the processing of bulk data and the extraction of salient details ("features.") Unlike file-based approaches, we have found that bulk data analysis is particularly well-suited to triage, as it allows for the rapid identification and extraction of features such as email addresses and credit card numbers. Experience has shown that the presence of such information is of significant value.

In this article we also present *bulk_extractor*, an open source tool that we have developed over the past two years, which embodies many of the discoveries that we have made in bulk data processing.

We have found that 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 identify which laptop or telephone has the largest collection of credit card numbers, allowing investigators to chose which disk to analyze first. In an organized crime or public corruption case, bulk data analysis can identify which hard drives or cell phones contain email addresses of interest: Bulk data analysis can even generate new leads, without the need for in-depth media analysis. Bulk data analysis further has the advantage of being *fast*: our experience has shown that our software typically processes digital media *ten times faster* than today's industry standard tools on hardware that is typical in most local or regional law enforcement agencies. Thus, systematic bulk data analysis has the potential for radically transforming the role of digital media analysis in law enforcement today, changing it from a conviction-support tool (through the identification of illegal materials) to assisting in investigations (through the identification of new leads and social networks).

1.1. Contributions

This paper makes several specific contributions to the theory and practice of digital forensics. First, this paper 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. We also show that bulk data analysis is easily adopted to stream processing and multithreading: our quantitative analysis shows linear speedup as additional cores are added to the analysis process.

We present *bulk_extractor*, a powerful tool for performing bulk data analysis. By sharing our experience in developing *bulk_extractor*, we present a model for the development of other digital forensic tools.

We introduce the concept of a context-sensitive stop list, and explain how stop lists that do not include local context are susceptible to manipulation by criminals

and other adversaries.

We present a new forensic disk image designed for testing the string recovery feature of modern computer forensic tools. We perform an evaluation using this test disk that compares our tool with Guidance Software's EnCase and the traditional approach of running the Unix *strings* and *grep* commands. In so doing, we show that our implementation can recover more kinds of high-value forensic features than current tools, and that it runs dramatically faster than current tools in both single-core and multi-core environments.

Finally, we provide two real-world case studies that show how features provided by bulk data analysis have been used productively early in a forensic investigation.

1.2. File-Based Forensics

Two approaches now dominate the processing of digital evidence: *file-based* and *bulk data* forensics.

File-based forensics is widely used by computer forensic examiners and implemented by popular tools such as Guidance Software's EnCase[25] and AccessData's FTK[4]. Such tools operate by finding, extracting, identifying and processing *files*—that is, a sequential collection of bytes and metadata (*e.g.* file name and timestamps).

Because it mirrors the way that users interact with computers, file-based forensics has the advantage of being easy to understand. The file-based approach also integrates well with most legal systems, because extracted files can be printed and entered into evidence. It has the disadvantage of ignoring data not contained within files, as well as ignoring the unallocated regions and metadata *within* compound document files not made visible by forensic tools. Additional limitations can be found in [18].

1.3. Bulk Data Analysis

In *bulk data analysis*, digital content is examined without regard to partitions or file system metadata. Instead, data of potential interest is identified by content and processed, extracted, and reported as necessary. File carving (§2.3) is an example of bulk data analysis.

An important aspect of bulk data analysis is that it can be applied 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.

As the forensic analysis of different parts of the bulk data can be conducted independently of each other, bulk data analysis is easier to parallelize than traditional forensic approaches that require reconstruction of the media's file system.

1.4. bulk_extractor: An Overview

bulk_extractor is a command-line program that extracts email addresses, credit card numbers, URLs, and other types of information from digital evidence files. *bulk_extractor* operates on disk images in raw, split-raw, EnCase E01[30], and AFF[23] formats. The disk image is split into *pages* and processed by one or more *scanners*. Identified features are stored in *feature files*, a simple line-based format that can be viewed directly with a text editor or processed by subsequent tools. The program produces report files containing extracted features, their location and frequency. The names of the files containing each feature can be determined through post-processing.

bulk_extractor detects and opportunistically decompress data in ZIP, gzip, and Microsoft's XPress Block Memory Compression algorithm[47]. This has proven useful, for example, in recovering email addresses from within fragments of corrupted system hibernation files.

bulk_extractor gets its speed through the use of compiled search expressions and multi-threading. The search expressions are written as pre-compiled regular expressions, essentially allowing *bulk_extractor* to perform searches on disparate terms in parallel (§3.2). Threading is accomplished through use of an *analysis thread pool* (§4.4).

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.

1.5. Outline of This Article

This concludes the introduction. Section 2 discusses related work. Section 3 discusses the design of our bulk analysis system and presents the design choices that were made. Section 4 discusses the actual implementation in *bulk_extractor*. 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. Recursive Re-Analysis and the Recovery of Compressed Information

The approach of recursively re-analyzing data encountered in forensic analysis is widely used by anti-virus programs, existing media forensics tools (*e.g.* [25, 4], and network forensics tools[13]. However, these tools generally apply re-analysis solely to intact files identified as being compressed or container files.

PyFlag[11] applies recursive reanalysis to data in runs of unallocated sectors, but only attempts to decompress at the beginning of each run. (The forensic path used by *bulk_extractor* to report the location of extracted features (§3.4) is similar to the virtual path reported by PyFlag.) The NetIntercept[13] network forensic tool automatically decompresses compressed network streams, but only for protocols known to use compression.

Our experience with bulk data analysis indicates that it is important to detect and decompress compressed data before performing feature extraction. We have identified 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 in the web cache. (The percentage of web servers employing compression increased from less than 5% to 30% between 2003 and 2010[39] because compression significantly increases the web performance[38, 46].)
2. The *.docx* and *.pptx* file formats used by Microsoft Office store content as compressed XML files in ZIP archives[19].
3. 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.
4. The Windows hibernation file is lightly compressed using a lightweight proprietary compression system[47].
5. Files are increasingly bundled together and distributed as ZIP, RAR, or *.tar.gz* archives for convenience and to decrease bandwidth requirements.

Today's forensic applications generally process compressed data when it is present in files, but do not proactively detect and decompress compressed data present in unallocated sectors, a process we call *opportunistic decompression*. We are aware of no other research or commercial tool that performs compression detection and opportunistic decompression for every byte of the subject media.

Although it is widely known that Windows hibernation files contain potentially valuable information, the beginning of each file is overwritten when Windows resumes from hibernation. We know of no other generally available tool that can use these corrupted hibernation files. For example, Zhao and Cao reported that they were unable to analyze the contents of hibernation files due to the use "an undocumented, Microsoft proprietary file format, including proprietary compression."[52]

2.2. Single-Block Forensics

In prior research we introduced a simplified form of bulk data processing that performed analysis of single sectors[**?**]. 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 our prior work did not.

2.3. Carving

Carving in digital forensics refers to the practice of extracting information based on content, rather than by relying on metadata that points to the content. *File carving* is a special kind of carving in which files are recovered. Although the two terms are frequently used interchangeably, *carving* describes a more general technique. For example, it is possible to carve a single still image out of a (possibly corrupt) video file[45].

File carving is useful for both data recovery and forensic investigations because it can recover files when sectors containing file system metadata are either overwritten or damaged. The earliest file carvers (*e.g.* [35, 42]) employed simple *header/footer carving*. Second-generation carvers (*e.g.* [17, 21]) perform this verification automatically, significantly reducing the amount of data produced. Some carvers (*e.g.* [17, 24]) can perform limited reconstruction of fragmented files.

2.4. 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. Previously we adopted the term *forensic feature extraction* to describe this process[20].

Today 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.* [5], [9, p. 304–305]). Credit card number searches are also performed by the Cornell University Spider forensic tool[14]. The University of Michigan Office of Information Technology Security Services reviewed several tools for discovering credit card and social security numbers in 2008[49].

The work presented here improves prior work in forensic feature extraction by using hand-tuned rules that consider local context to improve the precision of extraction without adversely impacting recall.

2.5. Parallel Processing

More than four decades of research have developed numerous approaches for speeding text processing through the use of parallelism. Indeed, Bird *et al.* discussed parallelized searching of a 50GB database *in the 1970s* using specialized machines[8]. Similar efforts continue to this day[31].

Although there have been some attempts in the research community 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.* considered the use of GPUs for speeding hashing to enable hash-based data carving[12], but no commercial forensic tool uses GPUs for hashing due to I/O limitations. Urias *et al.* considered a variety of issues involving the parallelized processing of RAID storage systems[2], but despite increasing media sizes, tools such as EnCase and FTK still are largely single-threaded.

The parallelization in this paper is distinguished from prior efforts in that we present a system for parallel processing that is easily implemented and automatically takes advantage of general-purpose multi-core CPUs, the most common form of parallelized hardware available. As such, this is parallelism that can be readily used by current practitioners.

2.6. 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 error rates that can be measured[50]. But little is known about the accuracy of feature extraction and string search of today's forensic tools. The National Institute of Standards and Technology's Computer Forensics Tool Testing Program created draft specifications have been written for String Search[40] and Deleted File Recovery[1], 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.* advocate testing tools with simplified data sets containing known targets[26]—for example, by validating string search creating documents containing the word "evidence" and then searching for it. We present an improvement to this procedure that involves embedding different targets in different file formats. This improvement makes it more complicated to prepare the test media, but also makes it significantly easier to determine the source of each extracted feature, since each feature is distinct.

3. Design of *bulk_extractor*

3.1. Requirements Study

Between 2003 and 2005 we developed a prototype bulk media analysis tool to assist in an unrelated investigation. We soon discovered that our tool was taking a fundamentally different approach than existing tools. Not only did the tool run

dramatically faster than file-based systems, but it became easy to answer questions of specific interest, such as *Does this drive contain sensitive information?* and *Who was the primary user of this drive?*

Because our tool did not easily map to existing categories, we conducted a series of unstructured interviews with local, state and federal law enforcement officers to determine if there was a need for this new kind of tool. In total, approximately 20 interviews took place between 2005 and 2008.

Analysts were generally enthusiastic about the research. They made specific requests that the tool 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

These interviews also provided us with a number of operational requirements for the intended tool:

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

We also learned from our interviews that the primary use of the tool would be for triage—that is, to prioritize which pieces of digital evidence should be analyzed first—to identify specific email addresses for follow-up investigation. Final analysis, however, would typically be performed with an "approved" tool.

The requirements analysis process was both informative and beneficial to our research and tool development.

3.2. Forensic Scanners and Feature Extractors

Our basic design uses multiple *scanners* that run sequentially on raw digital evidence. These scanners are provided with a buffer to analyze (initially corre-

sponding to a 16MiB 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. All scanners process all buffers 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*, also known as *feature extractors*, 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. Extracted features are recorded in a *feature file* (Figure 1).

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.

Forensic programs that process compressed data must guard against *decompression bombs*—files that, when fully decompressed, extend to many terabytes, petabytes, or even more[6]. We implement two 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).

3.3. Feature Files

In our interviews, analysts repeatedly requested that our tool provide output as a simple text file that could be viewed with an editor or processed with other "scripting" tools. To accommodate this request we designed the *feature file format*, a simple tab-delimited text file containing the offset at which the feature was found, the feature itself, and a configurable number of bytes that precede and follow the feature in the evidence file (Figure 1). Feature files are not sorted but are loosely ordered.

Although this format has proved useful, in some cases it is necessary to report multiple values associated with each extracted feature. In these cases we replace the third field with an XML fragment. For example, our design uses a block of XML to report all of the fields associated with EXIF structures found within embedded JPEGs. XML is useful because different EXIF structures contain different fields. A post-processing program transforms this XML-based feature file into a single CSV file that can be readily imported into Microsoft Excel.

Another post-processing tool can add a fourth column: the file from which the feature is extracted (see §3.4).

3.4. 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.

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 then store the compressed stream in the browser cache. In this case it is not enough to simply 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 reveal them: it is also necessary to explain *how* the data must be transformed to make them intelligible.

We resolve this problem by reporting 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 Figure 1 indicates that *jbarnes@virtuousgeek.org* is found by decompressing the GZIP stream found at 318707924 within the disk image; the email address occurs 70 characters 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 Figure 1 is found by decompressing the GZIP stream that begins at byte offset 946315592; 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/*.) We have found a high number of such double-compressed artifacts in the unallocated regions of subject media.

Forensic paths can be readily translated to a specific location in a resident or deleted file with a file system map. Our program *fiwalk*[**?**] 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 needs to be examined. The program *file_locations.py*, included with *bulk_extractor*, can then be used to annotate a feature file with the file names from the subject media corresponding to the sectors from which the features were extracted.

3.5. Histogram Processing

In previous work we showed that frequency distribution histograms to be of significant use in forensic investigations[20]. 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. Our design further allows histograms to be generated from substrings extracted from features using regular expressions. For example, this regular expression creates a histogram of search terms provided to Google, Yahoo, and other popular search engines:

```
search.*[?&/;fF][pq]=([^&/]+)
```
(1)

Histograms of search terms are particularly useful when conducting an investigation, as they reveal the intent of the computer's user, as we discuss in Section 6.1.[1]

3.6. 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, in our previous work we identified the email address *mazrob@panix.com* as being part of the Windows 95 Utopia Sound Scheme[20]. We suggested that one way to suppress these common features was to weight each feature by the inverse frequency with which the feature appears in the corpus, apparently an application of the well-known TF-IDF approach used in information retrieval[28].

For reasons that we did not anticipate in 2005, but which became clear during our interviews, it is not possible for many organizations to create a single list of all the email addresses extracted from every processed disk. Instead, many organizations manually produce lists of email addresses and domain names based on examiner experience that are ignored, technically known as *stop lists*.

Stop lists can be readily produced from default installs of popular operating systems. We have done this and found a staggering number of email addresses and URLs in some OSes. For example, Fedora Core 12 contains nearly 14 thousand distinct email addresses (see 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

[1] 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 murdered[3].

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.

Our solution is the introduction of *context-sensitive stop lists*. 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 creating a stop list containing just the email addresses found in default operating system installs, we create 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 the 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.

Finally, through use in actual cases have learned that items on a stop list should not be suppressed from the examiner. Instead, we report these items to separate 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.

4. Implementation

We have created *bulk_extractor*, a command-line program that implements our design and has been deployed to a number of production environments around the world. Version 1.0.0 of *bulk_extractor* consists of 8436 lines of C++ code and 1011 lines of GNU flex code.

4.1. Functional Modules

Our current implementation consists of five modules:

1. An **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 16MiB pages, and passes each page off to a thread in the thread pool.
3. The **analysis thread pool** operates 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 identify features that can be recovered.

5. The **feature recording module** records features identified by the scanners in one or more feature files.

This modular design makes it straightforward to add additional processing capabilities.

4.2. Scanner Implementation

We use purpose-built scanners based on regular expressions and hand-tuned rules to extract forensic information, an approach validated in the 1990s by natural language researchers[37]. The email scanner and the accounts scanner are both implemented as a series of regular expressions compiled using GNU flex [48] (Figure 2). This technique creates object code that can be quite large (the email scanner is 4 MiBytes) but runs quickly. Another scanner implements the AES key-finding algorithm developed by Halderman *et al.* [27]. The AES scanner is quite slow and is therefore disabled by default.

Some of our 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, while EnCase 6.2.1 identifies the string `Box [-568 -307 2000 1006] /FontName` as containing the US phone number `307 2000`, our system does not, as we recognize that the numbers `307 2000` are part of a larger group that does not conform to the US phone number pattern.

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 to the general comunity.

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

4.3. The Margin

It is straightforward to process a disk image block-by-block or page-by-page. Occasionally important features cross block or page boundaries. Many programs that process bulk data simply discard features that cross these block boundaries. In our interviews we learned that analysts are generally unaware that their tools systematically discard such features: upon learning of the behavior, they pronounced it unacceptable.

Our approach is to append to each buffer a *margin* of additional bytes from the following region of the evidence file. The implementation thus maintains two lengths for each buffer: the *buffer size* and the *margin size*. The margin is large

enough so that any feature or compressed region that begins near the end of the current page will fit within the margin. Only features that extend into the margin are reported; those that begin in the margin are suppressed and reported later, when the contents of the margin are re-processed.

We have experimentally determined that a margin of 1MiB is sufficient for most situations.

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 collections and to make the most efficient use of modern hardware (*e.g.* [41, 7]). But 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. Our approach of treating the disk image as a series of independent pages turns feature extraction into an "embarrassingly parallel" problem, as each page can be processed independently once features that cross page boundaries are properly handled (§4.3).

We devised three strategies for processing the pages in parallel:

1. **Regions:** The disk is divided into *N* equal regions, one for each thread.
2. **Striping:** Page 0 is processed by thread 0, page 1 by thread 1, and so on, up to page *N*, which is again processed by thread 0.
3. **Thread Pool:** One thread reads all of the pages in order and processed by a pool of worker threads.

We implemented all three strategies and found that the thread pool consistently outperformed the others with our test data downloaded from the `http://digitalcorpora.org` website.

We performed a series of tests to determine the success of our multi-threading approach and the relative performance of multiple threads under Windows and Linux. We used the 2.1GB *nps-2009-ubnist1* disk image[22]. The test machine was a MacBook Pro 8GiB of RAM and four physical cores (8 with hyperthreading) and an SSD, rebooting computer between each trial to assure that the disk was not cached the system's RAM. We tested under both the 64-bit MacOS 10.6 and Windows 7 operating systems.

With a single thread, MacOS required on average 3998 seconds to analyze the disk image, for an average performance of 0.53 MBytes/sec. With four analyzing threads the performance increased to 2.5 MB/sec—roughly a linear speedup, as evidenced by the first part of the graph. 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, we no longer see 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/sec. 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.

As can also be seen from the graph, adding additional analyzing threads resulted in lower performance, presumably due to increased issues. 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 we also saw linear speedup under Windows, the peak performance was roughly two-thirds that of the same hardware under MacOS. We cannot explain the poor performance of the Windows operating system, but note such observations are commonplace.

For a realistic comparison with EnCase, we used the 40GB *nps-2009-domexusers* disk image as our test data and a typical examiner's machine, a dual-processor Xenon X5650 at 2.67Ghz (12 physical cores, 12 hyperthreaded cores), with 12GB RAM running Windows 7 Professional. Our test was simple: extract and report all of the email addresses in the disk image. The results (Figure 4) indicate that our multi-threaded *bulk_extractor* extracted email addresses from the forensic test disk image 10x faster than EnCase 6.2.1 running on the same hardware, and was only 5% slower than simply running *ewfexport*, *strings* and *grep* (Figure 4). As we show in §5.1, *bulk_extractor* also finds significantly more email addresses than either EnCase or *strings*.

5. Validation

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

5.1. Constructed Test Drives

Although the recovery of email addresses from disk drives is a common forensic task, we could find no test procedures or data sets to help us compare *bulk_extractor*'s effectiveness with other tools. For example, the one NIST test disk for performing string search ([34]) only tests the ability to search for Cyrillic names, not email addresses. Meanwhile, the data sets created by Guo *et al.* for testing string search functions[26] have not been publicly released.

Faced with this lack of test images, we constructed our own. We created a series of test documents using various office productivity applications. Each document contained a single descriptive email address. In some cases we used the

applications to produce PDF files. All of the documents were stored in a disk image. The choice of document formats was based on our experience with real-world investigations. These documents were then stored in a FAT32 disk image and subjected to analysis by *bulk_extractor* and EnCase 6.2.1. We also analyzed the disk image with *strings* and *grep* as a control. Table 4 summarizes the results.

For every target *bulk_extractor* finds more email addresses than either EnCase or the Unix tools, primarily because of its ability to detect and recursively reanalyze compressed data. The only email addresses that *bulk_extractor* fails to find are email addresses in PDF files generated by Microsoft Office. Analysis revealed that while Apple's TextEdit preserves many strings (Figure 5) in the PDF file, Microsoft Word does not (Figure 6). Carrier has stated[10] that the only reliable way to extract text from PDFs is to render the page and process it with an optical character recognition program. Unfortunately, this approach is currently not possible with PDF fragments.

5.2. Base OS Installs

It is important to test forensic tools with base installations of operating systems. By characterizing the tool's behavior on base operating system installations we can gain insight as to how the tool will behave on actual data.

We used *bulk_extractor* to analyze default installations of nine operating systems, including two versions of Linux and five versions of Windows.

The Linux systems had thousands of domains, email addresses, URLs, telephone numbers, and components of ZIP files. Inspection revealed 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 were from software license agreements. For example, the most common phone number (found 47 times) in the Fedora release was (412) 268-4387, the number of the Carnegie Mellon University office of Technology Transfer. The most common false positive matches for telephone numbers came from arrays in PDF files, which sometimes were grouped as 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` solved this problem.

The Windows operating systems were, comparatively speaking, quite clean. We did not see a large collection of email addresses on the Windows installations. We did, however, discover a significant number of "IP addresses" that were in fact version numbers and SNMP OIDs.

5.3. Prevalence of Compressed Email Addresses

To gauge the value of *bulk_extractor*'s data decompressing feature, we analyzed disks and USB storage devices, purchased on secondary markets around the world, for the presence of email addresses that could be recovered using GZIP or ZIP decompression and were not otherwise present on the disk.

The disk images, taken from the Real Data Corpus,[22] came mostly from China, India, Israel and Mexico. To gauge the age of each drive we analyzed the Date: headers in email messages found on the disks. (This information is extracted into *bulk_extractor*'s *rfc822.txt* feature file.) We took the date of the disks' last activity by averaging the last five timestamps that were present. We manually reviewed a sampling of the timestamps to assure ourselves that they did in fact correspond to actual times.

For each drive we listed the email addresses and noted whether each address was present in plaintext or in a compressed stream. We then tallied the number of email addresses that appeared on each drive and on no other drive in our corpus (the same elimination approach that we proposed in [20]).

In total we analyzed 1473 drives. We found that 865 contained email addresses, of which 431 also contained timestamps. Table 5 tallies by year the number of drives recovered for that year, the total number of email addresses, the number of email addresses encoded with each compression format, and the number of email addresses only present on a single drive.

As can be seen, there are a significant number of email addresses that can *only* be recovered by opportunistically decompressing forensic data, demonstrating the importance of this technique.

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."[33] The day before the suspects' preliminary hearing, an evidence technician at the SLO police department was given a 250GB hard drive that had been seized with the suspects. 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 suspects lacked the necessary skills to commit the crime.

An early version of *bulk_extractor* was able to analyze the hard drive in roughly 2.5 hours. The email address and credit card number extractors, in combination with the histogram analysis, 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—information that could be used to create fraudulent credit cards. This helping to establish the defendant's intent[33].

Based on the reports generated by *bulk_extractor* and the testimony of the evidence 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.

6.2. ATM Fraud

A 250GB disk drive was recovered from individuals suspected of setting a credit-card "skimmer" and pinhole camera 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 shut down.

bulk_extractor completed its processing after just two hours 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. The banks were then able to immediately shut down the accounts before fraud could be committed. The actual files containing the data were later identified by using the file offsets present in the feature file.

As in the previous case, it is unlikely that today's commercial tools could have found the compromised credit card numbers in so short a time.

7. Limitations and Future Work

Bulk data analysis and *bulk_extractor* are designed to complement conventional forensic tools, not to replace them. In this section we discuss specific limitations that we have 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. Previous research has

shown that this is not a significant limitation; with the exception of log files, most files forensically interesting are not fragmented[21]. And while log files are fragmented, most are not stored with 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 reassembly[44].

7.2. Specific Design Limitations

Overall we have found *bulk_extractor*'s design to be quite powerful, as evidenced by the ease of adding new scanners and capabilities. Two limitations that have emerged, however, are the susceptibility to crashes caused from errors in C buffer processing, and the need to write individual scanners and decoders for each data type that we wish to recognize.

The issue of C buffer processing is not unique to *bulk_extractor*—this issue impacts every tool that is not coded in a typesafe language such as C# or Java. Rewriting *bulk_extractor* in such a language is complicated by the reliance on GNU flex and on libraries such as AFFLIB and libewf. We did create an earlier pure-Java version of *bulk_extractor* using JFlex[29] instead of flex. At the time we found that compilation with JFlex required minutes rather than seconds, but that the resulting scanner ran approximately three times faster—presumably a result of Java's optimizing just-in-time compiler. We are considering a rewrite of *bulk_extractor* into C# or Java.

7.3. Unicode and IDN issues

Unicode and Internationalized Domain Names (IDN) pose challenges for bulk data processing and extraction, as bulk data processing views data as a series of bytes but Unicode uses multiple bytes (and sometimes a variable number) for character encoding. 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. Disks from China or Japan may be coded in Big5, EUC-JP, GB18030, Shift-JIS, or other less popular coding schemes.

Fortunately, most email addresses and URLs encountered today are in simple ASCII, UTF-8 or UTF-16, all of which are handled by *bulk_extractor* to a varying degree using a simple but reasonably accurate algorithm: *bulk_extractor* contains regular expressions that recognize email addresses and URLs formed as 7-bit ASCII strings and as 8-bit strings where every other character is an ASCII NULL.

We are developing more sophisticated approach for text extraction. That approach is beyond the scope of this paper and will be discussed in a future publication.

7.4. SSDs and Other Random Access Media

The use of solid state drives (SSDs) and other forms of random access media is increasing. Modern SSDs are implemented using flash memory systems that support a limited number of writes to each cell. Media life is prolonged through the use of a flash translation layer (FTL) that maps the logical block addresses of the disk drive's interface to the physical pages that make up the storage system. Modern SSDs typically reserve 4–25% space beyond the rated storage capacity to support the FTL as well as previous versions of user data.

Recent experiments by Wei *et al.* [51] have shown that traditional file sanitization strategies of repeated overwrites do not work properly with SSDs. The authors demonstrated recovery by removing flash chips from a SSD and reading the data directly. The authors were not able to reconstruct the original file systems, however, as the operation of each drive's FTL is closely held proprietary corporate information.

Because it has the ability to recover useful investigative information without reconstructing individual files, in the future bulk data analysis may be an important tool in the analysis of SSDs.

7.5. Drive Profiling with Suppressed Terms

In §3.6 we presented our approach for suppressing commonly found features through the use of context-sensitive stop lists. Rather than being ignored, these features could be used for drive profiling, as their presence can be readily correlated with the installation and use of specific software.

8. Conclusion

This paper discusses a number of advances we have made in the field of bulk data analysis and presents the design and implementation of *bulk_extractor*, a forensic tool that extracts identifiable features such as email addresses, Internet URLs, and EXIF structures from bulk data. Information extracted by *bulk_extractor* is reported in *feature files* that indicate where each feature was found in the source file. *bulk_extractor* also performs histogram analysis, reporting, for example, the most common email addresses and search terms present on a hard drive. The tool automatically decompresses compressed data that it encounters.

bulk_extractor was developed as a research platform but has found use in some actual cases. Our users tell us that *bulk_extractor* is extraordinarily useful. We hope that interest in the tool and in bulk data analysis will continue to grow.

8.1. Code Availability

We have made the *bulk_extractor* source code, ancillary programs, and precompiled executables for Windows available on the *http://afflib.org/* website. The code is public domain and may be freely incorporated into other open source or commercial applications.

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

8.2. Acknowledgments

We gratefully acknowledge the participation of John Lehr at the San Luis Obispo Police Department for allowing us to print here the stories of the use of *bulk_extractor* with actual cases. Rob Beverly, Beth Rosenberg and others reviewed earlier versions of this paper and provided useful feedback. This work was supported by NSF Award DUE-0919593 and the Department of Defense.

References

[1] , 2005. Deleted file recovery tool specification. Tech. Rep. Draft for SC Review of Version 1.0, National Institute of Standards and Technology.
URL http://www.cftt.nist.gov/DeletedFileRecovery.htm

[2] , 2008. Consideration of issues for parallel digital forensics of raid systems. Journal of Digital Forensic Practice 2, 196–208.

[3] , Jun. 19 2008. Testimony expected on Neil Entwistle's computer activity the day of the murders of his wife, baby. Associated Press.
URL http://www.foxnews.com/story/0,2933,368604,00.html

[4] Access Data, 2005. Forensic toolkit—overview.
URL http://www.accessdata.com/Product04_Overview.htm?ProductNum=04

[5] AccessData Corporation, 2008. AccessData supplemental appendix: Regular expression searching.

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

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

[8] Bird, R. M., Tu, J. C., Worthy, R. M., January 1977. Associative/parallel processors for searching very large textual data bases. SIGMOD Rec. 9, 8–9.
URL http://doi.acm.org/10.1145/965645.810247

[9] Bunting, S., 2008. EnCE: The Official EnCase Certified Examiner Study Guide. SyBex.

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

[11] Cohen, M., Aug. 2008. PyFlag: an advanced network forensic framework. In: Proceedings of the 2008 Digital Forensics Research Workshop. DFRWS, [Online; accessed 06 March 2009].
URL http://www.pyflag.net

[12] Collange, S., Daumas, M., Dandass, Y. S., Defour, D., 2009. Using graphics processors for parallelizing hash-based data carving. In: Proceedings of the 42nd Hawaii International Conference on System Sciences.
URL http://hal.archives-ouvertes.fr/docs/00/35/09/62/PDF/ColDanDauDef09.pdf

[13] Corey, V., Peterman, C., Shearin, S., Greenberg, M. S., Van Bokkelen, J., 2002. Network forensics analysis. IEEE Internet Computing 6 (6), 60–66.

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

[15] Deutsch, L. P., May 1996. RFC 1951: DEFLATE compressed data format specification version 1.3. Status: INFORMATIONAL.

[16] Deutsch, L. P., May 1996. RFC 1952: GZIP file format specification version 4.3. Status: INFORMATIONAL.

[17] Digital Assembly, 2010. Adroit photo forensics.
URL http://digital-assembly.com/

[18] Garfinkel, S., 2010. Digital forensics: The next 10 years. Digital Investigation 7.

[19] Garfinkel, S., Migletz, J., Mar. / Apr. 2009. New XML-based files: Implications for forensics. IEEE Security & Privacy Magazine 7 (2).

[20] Garfinkel, S. L., Aug. 2006. Forensic feature extraction and cross-drive analysis. In: Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS). Elsevier, Lafayette, Indiana.
URL http://www.dfrws.org/2006/proceedings/10-Garfinkel.pdf

[21] Garfinkel, S. L., Aug. 2007. Carving contiguous and fragmented files with fast object validation. In: Proceedings of the 7th Annual Digital Forensic Research Workshop (DFRWS). Elsevier, Pittsburgh, PA.

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

[23] Garfinkel, S. L., Malan, D. J., Dubec, K.-A., Stevens, C. C., Pham, C., Jan. 2006. 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.

[24] Grenier, C., 2011. Photorec.
URL http://www.cgsecurity.org/wiki/PhotoRec

[25] Guidance Software, Inc., 2007. EnCase Forensic.
URL http://www.guidancesoftware.com/products/ef_index.asp

[26] Guo, Y., Beckett, J. S. J., 2010. Validation and verification of computer forensic software tools—searching function. Digital Investigation 6, S12–S22.

[27] Halderman, J. A., Schoen, S. D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J. A., Feldman, A. J., Appelbaum, J., Felten, E. W., May 2009. Lest we remember: cold-boot attacks on encryption keys. Commun. ACM 52, 91–98.
URL http://doi.acm.org/10.1145/1506409.1506429

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

[29] Klein, G., 2009. Jflex - the fast scanner generator for java.
URL http://jflex.de

[30] Kloet, B., Metz, J., Mora, R.-J., Loveall, D., Schreiber, D., 2008. libewf: Project info.
URL http://www.uitwisselplatform.nl/projects/libewf/

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

[32] Laboratory, R. C. F., 2008. Annaul report for fiscal year 2008.

[33] Lehr, J., Jun. 3 2010. Personal communication.

[34] Lyle, J., 2008. Unicode string searching—russian text.
URL http://www.cfreds.nist.gov/utf-16-russ.html

[35] Mikus, N., Kendall, K., Kornblum, J., Jan. 2006. Foremost(1). [Online; accessed 06 March 2009].
URL http://foremost.sourceforge.net/foremost.html

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

[37] Nadeau, D., Sekine, S., 2007. A survey of named entity recognition and classification.
URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.120.3657

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

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

[40] Program, C. F. T. T., Jan. 24 2008. Forensic string searching tool requirements specification.
URL http://www.cftt.nist.gov/ss-req-sc-draft-v1_0.pdf

[41] Richard, III, G. G., Roussev, V., 2006. Next-generation digital forensics. Commun. ACM 49 (2), 76–80.

[42] Richard III, G., Roussev, V., Aug. 2005. Scalpel: A frugal, high performance file carver. In: Proceedings of the 2005 Digital Forensics Research Workshop. DFRWS, New York.
URL http://www.digitalforensicssolutions.com/Scalpel/

[43] Secretary of State for the Home Department, Sep. 2007. 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.
URL http://www.official-documents.gov.uk/document/cm72/7215/7215.pdf

[44] Sencar, H., Memon, N., 2009. Identification and recovery of jpeg files with missing fragments. In: Digital Investigation. Vol. 6.
URL http://www.dfrws.org/2009/proceedings

[45] Snel, E., 2010. NFI defraser.
URL http://sourceforge.net/projects/defraser/

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

[47] Suiche, M., 2008. Windows hibernation file for fun 'n' profit. In: Black Hat 2008.
URL http://www.blackhat.com/presentations/bh-usa-08/Suiche/BH_US_08_Suiche_Windows_hibernation.pdf

[48] The Flex Project, 2008. flex: The fast lexical analyzer.
URL http://flex.sourceforge.net/

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

[50] US, 1993. Daubert v. Merrell Dow Pharmaceuticals. 509 US 579.

[51] Wei, M., Grupp, L. M., Spada, F. E., Swanson, S., 2011. Reliably erasing data from flash-based solid state drives. In: 9th USENIX Conference on File and Storage Technologies.

[52] Zhao, Q., Cao, T., Jan. 2009. Collecting sensitive information from windows physical memory. Journal of Computers 4 (1).

Tables

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

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.

VM	CCNs	Domains	Email	Exif	RFC822	Tel.	URLs	ZIPs
fedora12-64	1	13,973	21,612	119	2,017	662	75,555	55,172
macos10.6	35	2,432	2,669	909	485	256	6,781	55,793
redhat54-ent-64	0	12,345	17,669	36	2,052	3,773	25,078	20,749
win2003-32bit	0	475	227	6	41	65	7,878	149
win2003-64bit	0	330	172	5	40	42	7,421	163
win2008-r2-64	64	565	254	37	105	77	8,196	91
win7-enterprise-32	68	699	365	149	110	77	6,800	91
win7-ultimate-64	68	677	371	145	100	78	6,606	105
winXP-32bit-sp3	0	492	306	7	61	132	8,916	296
winXP-64bit	0	404	262	17	68	54	7,869	296

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.

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_find	User-provided regular expression searches.
scan_exif	JPEG EXIF headers
scan_kml	KML file recovery
scan_net	IP and TCP packets in virtual memory
scan_wordlist	Words (for password cracking)
Recursive Scanners:	
scan_base64	BASE64 coding
scan_gzip	GZIP[16] compressed files (including HTTP streams)
scan_hiberfile	Windows hibernation file decompression
scan_pdf	DEFLATE[15] compressed streams in PDF files and text extraction
scan_zip	Components of ZIP compressed files

Table 3: Scanners currently part of *bulk_extractor*.

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			✔

Table 4: We stored email addressed in sample documents using specific applications, placed the files in a disk image, and attempted to recover the email addresses using *strings* and *grep*, EnCase, and *bulk_extractor*. This table shows which email addresses could be recovered.
.

		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	2,663	0	0	16	0
1993	2	33,352	0	0	255	20
1994	4	18,401	310	0	1,096	421
1995	13	76,469	53	14	1,353	71
1996	19	314,340	10	1	43,567	1,532
1997	16	755,597	276	86	31,057	1,066
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	3,615
2001	51	1,180,587	360	109	9,118	710
2002	46	6,336,891	414	57	341,810	9,178
2003	44	1,654,880	429	95	19,444	563
2004	49	2,746,356	1,088	222	30,705	9,989
2005	27	351,238	75	3	2,656	127
2006	26	326,480	56	7	1,370	40
2007	17	828,229	301	0	3,319	388
2008	4	799,127	9	2	2,171	59

Table 5: A study of 431 disk 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.

Figure Captions

Figure 1. Two excerpts from a feature file generated by processing the NPS disk image *ubnist1.gen3.E01*[22]. The first column is the byte offset within the 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.

Figure 2. These excerpts from *bulk_extractor*'s *scan_accts.flex* input file for GNU flex[48] 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.

Figure 3. Speed of *bulk_extractor* to process the 43GB disk image *nps-2009-domexusers* as a function of number of threads and threading model. Reference computer is a MacPro with 16 GiB 1066 MHz DDR3 and two 2.26 GHz Quad-Core Intel Xeon processors, for eight physical cores and 16 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. The striping model, meanwhile, shows inconsistent performance improvement as a result of I/O contention.

Figure 4. Guidance Software's EnCase 6.2.1 takes 7 hours, 8 minutes to extract email addresses and other information from the 40GB disk image *nps-2009-domexusers*. *bulk_extractor* performs the same task in 274 minutes in single-threaded mode, and in 44 minutes in multi-threaded mode. The combination of *ewfexport*, *strings* and *grep* runs fastest, in just 42 minutes, 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.

Figure 5. 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.

Figure 6. 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.

Figures

```
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
```

Figure 1: Two excerpts from a feature file generated by processing the NPS disk image *ubnist1.gen3.E01*[22]. The first column is the byte offset within the 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.

```
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;
}
```

Figure 2: These excerpts from *bulk_extractor*'s *scan_accts.flex* input file for GNU flex[48] 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.

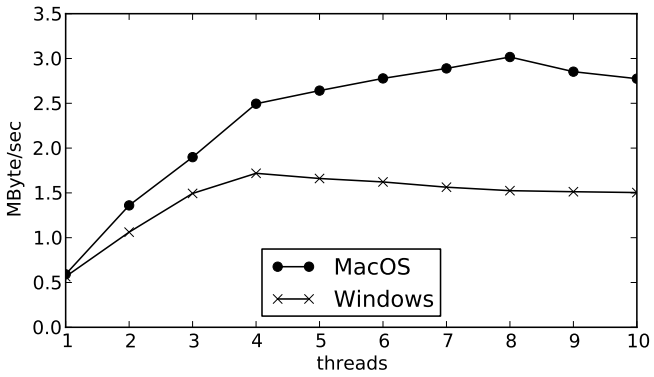

Figure 3: Speed of *bulk_extractor* to process the 43GB disk image *nps-2009-domexusers* as a function of number of threads and threading model. Reference computer is a MacPro with 16 GiB 1066 MHz DDR3 and two 2.26 GHz Quad-Core Intel Xeon processors, for eight physical cores and 16 virtual cores with 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. The striping model, meanwhile, shows inconsistent performance improvement as a result of I/O contention.

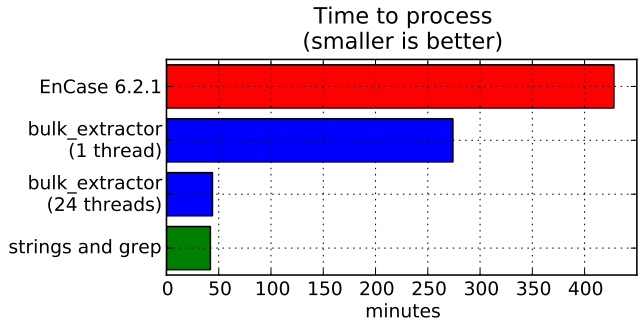

Figure 4: Guidance Software's EnCase 6.2.1 takes 7 hours, 8 minutes to extract email addresses and other information from the 40GB disk image *nps-2009-domexusers*. *bulk_extractor* performs the same task in 274 minutes in single-threaded mode, and in 44 minutes in multi-threaded mode. The combination of *ewfexport*, *strings* and *grep* runs fastest, in just 42 minutes, 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.

```
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
```

Figure 5: 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.

```
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
```

Figure 6: 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.