DRAFT
Digital media triage with stream-based forensics and *bulk_extractor*

Simson L. Garfinkel

***Abstract*—Stream-based forensics eschews file extraction and analysis, common in forensic practice today, and instead scans an entire disk image from beginning to end, extracting salient details that are of use in the typical digital forensics investigation. This article presents the requirements, design and implementation of *bulk_extractor*, a new, high-performance carving and feature extraction tool that uses stream-based forensic for triage and rapid analysis of digital media. *bulk_extractor* offers several important advances, including the optimistic decompression of compressed data, context-based stop-lists, and the creation of a *forensic path* that allows concise documentation of both the physical location and forensic transformations necessary to reconstruct exploited evidence. Although *bulk_extractor* was developed as a prototype, it has proved useful in actual police investigations, two of which we recount.**

I. Introduction

With each passing year the task of digital forensics investigators becomes more difficult as the capacity and variety of devices containing digital evidence continues to increase. A variety of approaches have been proposed for addressing the data onslaught, including parallelization and multi-processing[6], [43], the use of statistical sampling[17], [38], and even legislative solutions such as extending the amount of time that a suspect can be held without charges being filed so that evidence may be completely analyzed[46].

A. Contributions

This paper makes many specific contributions that further both the theory and the practice of digital forensics. First, this paper shows that bulk data analysis, and especially stream-based bulk data analysis, can productively be used as a fast analysis step during the early part of an investigation. We show that significant analysis can take place *during stream processing* to allow for both triage and for the rapid identification of important investigative leads. We present *bulk_extractor*, a powerful tool for performing stream-based bulk data analysis. By sharing our experience in developing *bulk_extractor*, we present a model for the development of other digital forensic tools. By their design, stream-based bulk data analysis tools are considerably easier to adapt to multi-core parallelization than traditional file-based forensic approaches: we demonstrate this claim with a quantitative analysis of the speedup provided through the use of multiple cores. We show that our implementation can recover more kinds of high-value forensic features than current tools, and that it runs faster in both single-core and multi-core environments. We also present a new forensic test disk explicitly designed to test forensic feature extraction, and show why our tool can find email addresses in the PDF files created by Apple TextEdit but not by Microsoft Word. Finally, we provide two real-world case studies that show how features provided by stream-based forensics can be used productively early in a forensic investigation.

B. Stream-Based Forensics

Stream-based forensics is an alternative data processing model for digital forensics that involves processing an entire disk image as a single bytestream, starting at the beginning and reading until the end. This approach largely eliminates the time that the drive head spends seeking between files and assures that no data in the disk image will be left untouched.

A second important aspect of stream-based forensics is that it can be applied to all kinds of computer systems, file systems, and file types—especially those that are not handled by existing tools. Stream-based systems can also be readily applied to media that has been damaged or partially overwritten.

Stream-based processing is far easier to parallelize than traditional forensic approaches, making it relatively easy to deliver software that fully utilizes today's multi-core microprocessors.

Our testing indicates that stream-based systems tuned to specific tasks of forensic import significantly outperform existing tools both in terms of the quantity of features extracted and the overall extraction speed. For example, our multi-threaded *bulk_extractor* can extract email addresses from a standard forensic test disk image 10x faster than EnCase 6.2.1 same hardware, and only 5% slower than simply running *ewfexport*, *strings* and *grep* (see Figure 1).

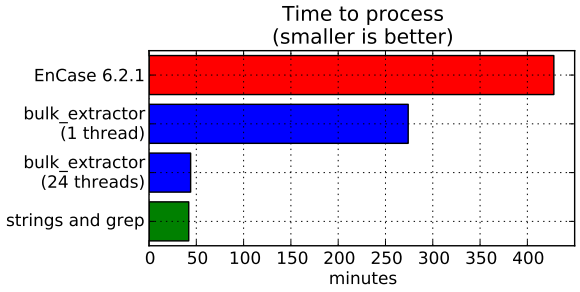

Fig. 1. Guidance Software's EnCase 6.2.1 takes 7 hours, 8 minutes to extract email addresses and other information from the 40GB disk image *nps-2009-domexusers*. *bulk_extractor* performs the same task in 274 minutes in single-threaded mode, and in 44 minutes in multi-threaded mode. The combination 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. Hardware is a dual-processor Xenon X5650 @ 2.67Ghz (12 physical cores, 12 hyperthreaded cores), 12GB RAM running Windows 7 Professional.

C. 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[32], and AFF[22] format. 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.

bulk_extractor detects and optimistically decompresses data in ZIP, gzip, and Microsoft's XPress Block Memory Compression algorithm[49]. This has proven useful, for example, for recovering email addresses from within fragments of corrupted system hibernation files. *bulk_extractor* contains a simple but effective mechanism for protection against decompression bombs[5].

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 (§III-B). Threading is accomplished through the use of an *analysis thread pool* (§IV-E).

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.

D. Outline of This Article

This concludes the introduction. Section II discusses related work. Section III discusses the design of our stream-based forensics system and presents the design choices that were made. Section IV discusses the actual implementation in *bulk_extractor*. Section V presents the results of *bulk_extractor* run against both test data and hundreds of forensic images. Section VI presents two case studies. Section VII discusses limitations of our architecture and opportunities for future work. Section VIII concludes.

II. Related Work

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

A. File-Based Forensics

File-based forensics are widely used by computer forensic examiners and implemented by popular tools such as EnCase[25] and FTK[3]. Such tools operate by finding, extract, identify and processing *files*—that is, a collection of bytes and metadata (*e.g.* file name and timestamps). While some tools can recover information from the unallocated sectors with *file carving*, many practitioners eschew carving due to the time required and the number of false positives generated.

File-based forensics has the advantage of being easy to understand because it mirrors the way that users interact with computers. The file-based approach also works well with the legal system, because extracted files can be printed and entered into evidence. It has the disadvantage of largely ignoring sectors that are not contained within files, as well as ignoring the unallocated regions and metadata *within* compound document files that are not made visible by forensic tools. Additional limitations can be found in [15].

B. Bulk Data and Stream-Based Forensics

In *bulk data forensics* digital content is examined without regard to partitions or file system metadata. Instead, data of potential interest is identified by content and processed, extracted, or reported as necessary. This is the approach used by file carvers (§II-D). Roussev suggested the term *stream-based forensics* to describe a particular form of bulk data processing in which a device is examined from the beginning to the end as a single stream for efficiency[45].

With others, we have previously introduced another bulk data approach based on random sampling of forensic data[17]. Random sampling is similar to the stream-based approach presented here, in that random sampling operates independently of the file system and other

metadata. Indeed, our random sampling system shares some code with the system presented here. But the approach present here is fundamentally different from the random sampling approach in that the entire evidence file is examined for the purpose of extracting all identity information that the media may contain. Further, the work presented here is more comprehensive than the work presented previously, as it includes recursive re-analysis and the use of feature extraction.

Outside of computer forensics there has been significant work on stream processing. For example, Hristidis *et al.* discuss an approach for performing continuous keyword queries on multiple text streams[28]. But an implicit assumption in their work is that the text streams will be well-formed text—for example, RSS feeds and blogs. They avoid one of the fundamental problem that we solve—the identification and extraction of text from an undifferentiated binary stream of digital evidence.

Likewise, Weinstein discusses in his PhD thesis a variety of problems that arise when searching large audio databases for speech and music[54]. Similar efforts are underway to build large, indexed archives of video that allow for efficient content-based retrieval[23]. Although these are important technologies that will be used by computer forensics practitioners, they assume a database of well-formed audio and video. The stream-based approach discussed here could be used to rapidly extract audio files from digital evidence and incorporate them into a database that could then be processed using such approaches.

C. 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 a 50GB database in the 1970s using specialized machines[7]. Similar efforts continue to this day[33]. The parallelization in this paper is distinguished from prior efforts in that we present the use of parallel processing for the extraction of useful information from digital forensic evidence: this is the first work that we are aware of that moves parallelism directly to the point of file and feature extraction from a digital evidence file.

D. 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[47].

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.* [36], [44]) employed simple *header/footer carving*. Second-generation carvers (*e.g.* [14], [19]) perform this verification automatically, significantly reducing the amount of data produced. Some carvers (*e.g.* [14], [24]) can perform limited reconstruction of fragmented files.

E. 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 kinds of artifacts can aid many investigations. Previously we adopted the term *forensic feature extraction* to describe this process[18].

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.* [4], [8, p. 304–305]). Credit card number searches are also performed by the Cornell University Spider forensic tool[11]. The University of Michigan office of Information Technology Security Services reviewed several tools for discovering credit card and social security numbers in 2008[51].

F. Recovery of Compressed Information

Our experience with bulk data forensics indicates that it is important to detect and properly process compressed data. 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[41] because compression significantly increases the web performance[40], [48].)
2) The new *.docx* and *.pptx* file formats used by Microsoft Office store content as compressed XML files in ZIP archives[16].
3) NTFS file compression may result in files being compressed. In practice, the files most likely to be compressed are Windows restore points.
4) The Windows hibernation file is lightly compressed using a lightweight proprietary compression system[49].

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. Two notable exceptions are PyFlag[9], which evaluates each deleted file to see if it is a gzip-compressed stream; and NetIntercept[10], which automatically decompresses compressed network streams. The forensic path used by *bulk_extractor* to report the location of extracted features (§III-D) is similar to the virtual path reported by PyFlag.

G. 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[52]. But to date, most of the tools tested by the National Institute of Standards and Technology's Computer Forensics Tool Testing Program have been disk imagers and write blockers. A draft specification have been written for String Search[42] and Deleted File Recovery[1], but the drafts have not advanced past an initial version and no test results have been published by NIST. Important real-world requirements are missing from these drafts, such as the ability to recover information from complex document files and compressed data.

In the absence of a standard, Guo *et al.* advocate testing tools with simplified data sets containing known targets. For example, Guo proposes a procedure for validating the "Searching Function" of forensic tools by hiding the word "evidence" in different kinds of documents and then seeing if the tool can recover the word[26]. We suggest an important modification to Guo's procedure is to hide different targets in different file formats, a process that makes analyzing test results considerably easier.

III. Design

A. Requirements Study

We conducted a series of unstructured interviews with local, state and federal law enforcement officers to determine if there was a need for a tool that could rapidly extract certain kinds of sensitive information from disk images. In total, approximately 20 interviews took place between 2005 and 2008. During these meetings we received specific requests for a tool that could automatically extract and create a report containing:

- 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.

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

- Able to run on Windows, Linux and Macintosh-based systems.
- Run with no user input.
- Operate on raw disk images, split-raw volumes, EnCase E01 files, and AFF files.
- 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.
- The tool should never crash.

B. Forensic Scanners and Feature Extractors

Our basic design uses multiple *scanners* that are run sequentially on a buffer of raw digital evidence. These scanners provided with a buffer to analyze (initially corresponding to a 16MiB page of data read from the disk image), the location or *path* of the buffer's first byte.

There are two kinds 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 kinds of recognizable strings that are likely to be in email messages. Extracted features are recorded in a *feature file* (Figure 2).

Recursive scanners, as their name implies, can decode data and pass it back to the buffer processor for reanalysis. 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 processes compressed data must guard against *decompression bombs* that, when fully decompressed, extend to many terabytes, petabytes, or even more[5]. There are two defenses against compression bombs. First, only a 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).

C. Feature Files

In our interviews we were repeatedly asked by analysts to provide output as a simple text file that could be

```
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 NPS disk image ubnist1.gen3.E01[21]. 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.

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 format. Each line consists of the offset at which the feature was found, the feature itself, and finally the context in which the feature was found (Figure 2).

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 have found it useful to replace the third field of the feature file 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 here because different EXIF structures contain different fields. A post-processing program can then transforms the feature file into a single CSV file that can be readily imported into Microsoft Excel.

D. 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 that is viewed using a webmail service. If the web client and server both support HTTP compression the web page will most likely be downloaded over the Internet and saved in the browser cache using GZIP compression. In this case it is not enough to simply report *where* the credit card numbers is found, because looking at the sectors in the disk image with a hex editor will only show a compressed binary stream: it is also necessary to explain *how* the data must be transformed to make it is viewable.

We resolve this problem by reporting a *forensic path* for each feature that is found. In most cases 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 2 indicates that the email address *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. For example, the email address *nelson@crynwr.com* in Figure 2 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 */var/cache/apt/archives/gnash-common_0.8.4-0ubuntu1_i386.deb*.) We have found a high number of such double-compressed artifacts in the unallocated regions of subject media. They have been ignored by the current generation of forensic tools because today's carvers do not optimistically decompress the data that they find.

Forensic paths can be readily translated to a specific location in a resident or deleted file with a file system map. Such a map can be formed as a Digital Forensics XML file that contains the name of every file and that file's location in the digital media. Our program *fiwalk*[20] 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.[1]

E. Histogram Processing

In previous work we showed that frequency distribution histograms can be of significant use in forensic investigations[18]. 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.

[1]The program *file_locations.py*, included with *bulk_extractor*, will rapidly annotate a feature file with the names of the files that correspond to the sectors from which the features were extracted.

Histogram generation can be efficiently integrated with the feature recording system, allowing histograms can be created for any feature. Our design allows histograms to be generated from substrings extracted from features using regular expressions. For example, the regular expression `search.*[?&/;fF][pq]=([^&/]+)` creates a histogram of search terms provided to Google, Yahoo, and other popular search engines. Histograms of search terms are particularly useful when conducting an investigation, as they reveal the intent of the computer's user.[2]

F. 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[18]. 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[30].

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 the email addresses that they have extracted from every disk that has been processed. Instead, many organizations manually produce *stop lists* of email addresses and domain names based on examiner experience. The email addresses in these stop lists are then summarily ignored.

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 II). Additional email addresses and URLs are present in application programs.

Clearly, a forensic analyst who does not employ stoplists will be overwhelmed by such information. However, we have determined that there is also a significant danger to 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 stoplists.

[2]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[2].

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 that contains just the email addresses that are found in default operating system installs, our context-sensitive stop list contains the local context.

Table I shows the results of histogram processing on the *nps-2009-domexusers* disk image before and after the application of the context-sensitive stop produced from several a default Windows XP, 2000 and 2003 installations. The stop list removes email addresses that are 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, we 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 included.

IV. 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. Currently *bulk_extractor* consists of 7664 lines of C++ code and 814 lines of GNU flex code.

A. 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** processes a 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. Feature files are tab-delimited text files that contain the forensic path, the feature, and the local context in which that feature was found (see Figure 2). Feature files are not sorted but are loosely ordered.

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

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 I
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
w2k3-32bit	0	475	227	6	41	65	7,878	149
w2k3-64bit	0	330	172	5	40	42	7,421	163
win2008-r2-64	64	565	254	37	105	77	8,196	91
win7-ent-32	68	699	365	149	110	77	6,800	91
win7-utl-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 II
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.

B. Scanner Implementation

We use purpose-built scanners based on regular expressions and simple rules to extract forensic information, an approach validated in the 1990s by natural language researchers[39]. The email scanner and the accounts scanner are both implemented as a series of regular expressions that are compiled using GNU flex [50] (Figure 3). 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].

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 filters that examine 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.

The current scanners incorporated into *bulk_extractor* are shown in Table III.

C. The Margin

It is straightforward to process a disk image block-by-block or page-by-page, but occasionally important features cross block or page boundaries. Many programs that process bulk data encounter this problem.

Our approach is to append to each buffer a *margin* of additional bytes that extend from the current page into the next. 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 starts near the end of the current page will be captured entirely 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.

```
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[50] 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.

Name	Recognizes
Basic Scanners:	
scan_accts	Credit card numbers, phone numbers, and other formatted numbers
scan_aes	Scans for AES key schedules in memory
scan_bulk	bulk data statistics
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[13] compressed files (including HTTP streams)
scan_hiberfile	Windows hibernation file decompression
scan_pdf	DEFLATE[12] compressed streams in PDF files and text extraction
scan_zip	Components of ZIP compressed files

TABLE III
SCANNERS THAT ARE CURRENTLY PART OF *bulk_extractor*.

D. Crash Diagnosis and Recovery

Developers of forensic software are plagued by the difficulty of diagnosing crashes. Unlike other types of software, forensic tools are invariably run on data that is incomplete or corrupt. Many crashes are data-dependent: tools that run reliably on hundreds of data sets and occasionally crash when presented with new data[37]. Replicating these crashes can be nearly impossible without the specific data at hand. Unfortunately, users frequently experience crashes caused by data that cannot be shared with the developers due to issues of privacy or confidentiality.

To address this problem we developed a detailed log file that records *what* operations were performed but which does not contain any forensic data. This log file is in ASCII and can be readily inspected for the presence of sensitive information. In the even of a crash the program can be re-run with logging enabled; the relevant lines can then be manually transcribed, retyped, or even read over the phone to a developer, who will then posses sufficient information to determine at least the location of the faulty code. This diagnostic proved useful in determining the causes of several crashes.

There were some cases, however, in which sufficient information was not present. For these cases we developed a *crash recovery mode*. As *bulk_extractor* runs, the primary thread writes a checkpoint to the output directory (the directory where all of the feature files are saved). In the event of a crash, *bulk_extractor* can be re-run with the same command arguments and it will then restart slightly beyond the point that caused the crash. In practice, analysts are told to simply hit the up-arrow and press Enter in the event of a crash, and *bulk_extractor*

continues from where it left off. Unfortunately, this crash recovery works so well that analysts no longer have an incentive to participate in debugging crashes.

To further protect *bulk_extractor* against crashes, a signal handler can be set before each scanner is called. The handler catches invalid memory references and returns control to the page processor. This works acceptably because most pointer errors in C/C++ forensic programs result in invalid pointer references on read, rather than on write.

E. 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.* [43], [6]). Tool developers nevertheless have been slow to adopt parallelized algorithms due to the complexity of doing so—specifically the problem of managing multiple analyzers and combining the results. Here we discuss alternative design strategies for parallelizing *bulk_extractor* and present the rationale for the decisions that we made.

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 (see Section IV-C).

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:** A master thread reads all of the pages in order and hands them off to a pool of worker threads $1, 2, 3, \ldots (N-1)$ as each thread becomes available.

We implemented all three strategies. We found that the regions strategy works well in a cluster environment where each node has its own copy of the forensic image. But in more typical environments where the evidence is stored on a single spindle, the regions approach exhibited a performance problem: because all of the threads run in parallel, this strategy resulted in excessive seeking of the disk drive head as each thread processed more information. We thus observed high seeking and relatively poor performance.

The striping and thread pool approaches have the advantage that the disk image is read from the beginning to the end. But we found that the striped threads invariably became desynchronized, resulting in the same kind of seeking overhead that happened with the regions approach (although not as severe). The desynchronization happened because different pages required different processing times. Attempts to keep the threads in sync would result in some threads idling until the slower threads caught up.

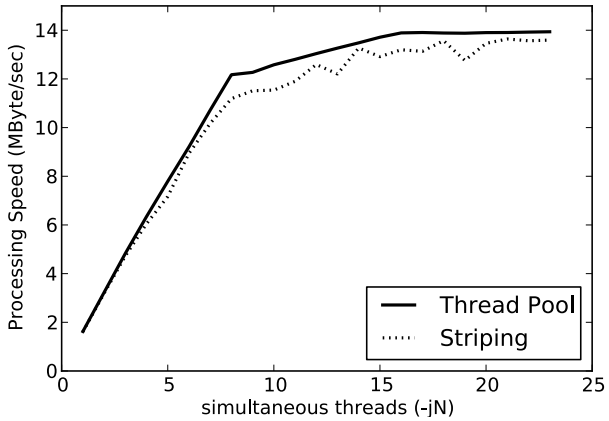

Fig. 4. Speed of *bulk_extractor* to process the 43GB disk image *nps-2009-domexusers* as a function of number of threads and threading model. Reference computer is a MacPro with 16 GiB 1066 MHz DDR3 and two 2.26 Ghz Quad-Core Intel Xeon processors, for eight physical cores and 16 virtual cores with 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.

The regions and striping approaches also suffered from the need to maintain separate feature files for each thread and then recombine the files when the threads had finished executing. This created considerable code complexity that was not initially anticipated.

The "thread pool" strategy was relatively simple to implement once the underlying single-producer/many-consumer mechanism was implemented. We used POSIX threads to provide multi-threading on Unix, and the *Pthreads-win32* package[29] on Windows.

We performed a series of performance tests using the *nps-2009-domexusers* disk image[21]. The test machine was a dual processor MacPro with 16GiB of RAM and eight physical cores (16 with hyperthreading). We used the raw 43GB image, large enough to assure that the disk image could not fit in the system's RAM. For each case three runs were performed; the fastest of the three runs are plotted in Figure 4. (We chose not to average the runs because slower speeds on a Macintosh are invariably the result of a background process over which we have no control.) The data was collected with a previous development version of *bulk_extractor* that supported both POSIX threads and striping.

The fastest raw run with a single thread was 26,481

seconds, resulting in an average data transfer speed of 1.6 MBytes/sec. The fastest run with eight analyzing processes was 3500 seconds, for a sustained processing speed of 12.2 MB/sec, a speedup of 7.6—nearly linear speedup, as evidenced by the first part of the graph.

Cores 9 through 16 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 9 to 16 is itself linear. With 16 processes the raw image was processed in 3092 seconds, for a sustained processing rate of 13.9 MB/sec, an overall speedup of 870%. Thus, the additional eight hyperthreaded "cores" provide roughly the speed of a 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 little available for the virtualized cores.

As can also be seen from the graph, the thread pool approach consistently provides greater performance than the striping. It also provides more consistent performance, presumably because the evidence file is now being read sequentially from beginning to end, eliminating much of the seeking.

V. Validation

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

A. 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 at these tasks with other tools. For example, the one NIST test disk for performing string search ([35]) 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 the 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 on a disk image which was then processed using *bulk_extractor*. The choice of document formats was based on our experience with real-world investigations. We then analyzed this disk image with three tools: Unix *strings*, EnCase, and *bulk_extractor*. 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 IV summarizes the results.


```
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. 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.

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* failed to find are email addresses in PDF files generated by Microsoft Office—somewhat curious, as *bulk_extractor* is able to extract email addresses from PDF files generated by Apple's TextEdit application. The performance difference is a result of differences in the internal structure of the PDF documents generated by the two applications: Apple's TextEdit tend to preserve strings (Figure 5), allowing the email addresses to be found when *bulk_extractor* decompresses the PDF streams and rescans them, while Microsoft Word tends to break up words inside the PDF stream(Figure 6), making feature extraction of some features impossible without additional processing.

B. 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. We then manually examined the results, both to improve the regular expressions and false-positive rejection logic, as well as to characterize the behavior of the tools.

We saw considerably different behavior between the Linux and the Windows systems. 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 li-

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 IV
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.

.

cense 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. Many of the remaining addresses were from certificate authorities or Microsoft-owned services such as *domain.microsoft.com* or *sign.msn.com*.

C. 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. We hypothesized that such data would be present, and that such email addresses would be more prevelant in recent drives and less so on older ones.

The disk images that we analyzed were part of the Real Data Corpus[21] and came mostly from China, India, Israel and Mexico. To gauge the age of each drive we analyzed the Date: headers in email messages and HTTP headers 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.

For each drive we listed the email addresses and noted for each address if it was present in plaintext, in GZIP or ZIP compressed streams, and in Base64 encoded streams. (The Base64 data is not reported here.) We then tallied the number of email addresses that appeared on each drive and on no other drive in our corpus; this was a straightforward approach for removing the email addresses that were part of operating system and application installs (although it assumes that each

```
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. 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.

operating system and application was installed on at least two disks in our corpus). This is the same elimination approach that we proposed in our previous work[18].

In total we analyzed 1473 drives. We found that 865 contained email addresses, 431 contained timestamps, and 431 contained both email addresses and timestamps. Table V 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 those 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 optimistically decompressing forensic data, demonstrating the importance of this technique.

VI. Case Studies

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

A. 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."[34] 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 associate of the defendants who had not been arrested, 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:

- 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 used to create fraudulent credit cards, helping to establish the defendant's intent[34].

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 such little effort on the part of the investigator) with a conventional forensic tool. While *bulk_extractor* can process a disk image at roughly the speed it takes to read the data (on a suitably fast machine), conventional tools such as EnCase or FTK can take anywhere from 2 to 10 times as long to ingest a disk image, a process that involves finding all of the files and indexing the data that they contain. Such tools also lack *bulk_extractor*'s histogram analysis and its ability to extract information from residual data that is compressed.

B. 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 `ccns.txt` output file, a list of credit-card numbers found on the drive. 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 the current commercial tools could have found the compromised credit card numbers in so short a time.

		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 V
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.

VII. LIMITATIONS AND FUTURE WORK

Stream-based forensics 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 discuss the outlook for this technology.

A. Theoretical Limitations of Stream Processing

The primary limitation of stream processing is that compressed objects fragmented across multiple locations cannot be recovered. Previous research has shown that this is not a significant limitation; with the exception of log files, most files that are forensically interesting are not fragmented[19]. And while log files are fragmented, most log files are not stored with compression, allowing the various fragments to be matched and recombined.

B. 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[31] 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.

C. Unicode and IDN issues

Unicode and Internationalized Domain Names (IDN) pose challenges for bulk data processing and extraction based on regular expressions, as bulk data processing views data as a series of bytes but Unicode combines multiple bytes to form characters.

Most modern text is encoded in either ASCII, UTF-8, UTF-16LE, or UTF-16BE. It can be difficult to distinguish these variant coding schemes. Although the Unicode Byte Order Mark (U+FEFF) is used to distinguish UTF-16LE from UTF-16BE, fragments of Unicode found within residual data may be missing the BOM.

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.

The best way to process bulk data is to scan the data and determine which coding, if any, is in use. Individual characters should be extracted, transcoded into Unicode if necessary, and then processed with a Unicode-aware lexical analyzer. The task is complicated by the fact that some bulk data may have multiple codings, and

that some codings cannot be inferred from context—for example, it is not possible to infer if text is coded in Code Page 737 (Greek) or Code Page 862 (Hebrew) without attempting to decode the text in both code pages and then applying a language model to determine which coding is more likely.

Fortunately, such approaches are sufficient for the majority of bulk data analysis today, as most email addresses and URLs 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. This handles both the case of 7-bit ASCII, UTF-8, and UTF-16 (both LE and BE) when it is used to encode ASCII strings.

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

D. SSDs and Other Random Access Media

At the present time 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% to 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.* have shown that traditional file sanitization strategies of repeated overwrites do not work properly with SSDs[53]. The authors demonstrated recovery by removing flash chips from a SSD and reading the data directly. The authors were not able to restore 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 the need reconstruct individual files, stream based forensics may be an important tool in the analysis of SSDs in the future. This procedure would be made easier through the use of vendor-proprietary commands that allow reading the raw flash pages present in each chip without the need to reverse engineer or otherwise understand the FTL.

VIII. Conclusion

This paper has introduced both stream-based forensics and *bulk_extractor*, a forensic tool that works instead on bulk data and extracts small-sized identifiable features such as email addresses, Internet URLs, and EXIF structures. 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 is encountered.

bulk_extractor was initially developed as a tool to assist in other research projects, and grew into a research interest in its own right. Today the tool is also being used to support law enforcement activities. We have found *bulk_extractor* to be extraordinarily useful, and hope that interest in the tool and in stream-based forensics will continue to grow.

A. Code Availability

We have made *bulk_extractor* source code, ancillary programs, and pre-compiled 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*.

B. 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* on actual cases. Rob Beverly, Beth Rosenberg and others reviewed earlier versions of this paper and provided useful feedback. We express our gratitude to the anonymous reviewers, especially to Reviewer #3, whose comments were invaluable in the maturation of this article. This work was supported in part by NSF Award DUE-0919593.

References

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

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

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

[4] AccessData Corporation. AccessData supplemental appendix: Regular expression searching, 2008. http://www.accessdata.com/downloads/media/Regular%20expression%20%207-24-08.pdf.

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

[6] Daniel Ayers. A second generation computer forensic analysis system. In *Proceedings of the 2009 Digital Forensics Research Workshop*. DFRWS, August 2009.

[7] R. M. Bird, J. C. Tu, and R. M. Worthy. Associative/parallel processors for searching very large textual data bases. *SIGMOD Rec.*, 9:8–9, January 1977. ISSN 0163-5808. http://doi.acm.org/10.1145/965645.810247.

[8] Steve Bunting. *EnCE: The Official EnCase Certified Examiner Study Guide*. SyBex, 2008.

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

[10] Vicka Corey, Charles Peterman, Sybil Shearin, Michael S. Greenberg, and James Van Bokkelen. Network forensics analysis. *IEEE Internet Computing*, 6(6):60–66, 2002. ISSN 1089-7801.

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

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

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

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

[15] Simson Garfinkel. Digital forensics: The next 10 years. In *DFRWS 2010*. Portland, OR, 2010.

[16] Simson Garfinkel and James Migletz. New XML-based files: Implications for forensics. *IEEE Security & Privacy Magazine*, 7(2), March / April 2009.

[17] Simson Garfinkel, Alex Nelson, Douglas White, and Vassil Roussev. Using purpose-built functions and block hashes to enable small block and sub-file forensics. In *DFRWS 2010*. Elsevier, Portland, OR, 2010.

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

[19] Simson L. Garfinkel. Carving contiguous and fragmented files with fast object validation. In *Proceedings of the 7th Annual Digital Forensic Research Workshop (DFRWS)*, August 2007.

[20] Simson L. Garfinkel. 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, IEEE, Oakland, CA, 2009.

[21] Simson L. Garfinkel, Paul Farrell, Vassil Roussev, and George Dinolt. Bringing science to digital forensics with standardized forensic corpora. In *Proceedings of the 9th Annual Digital Forensic Research Workshop (DFRWS)*. Elsevier, London, August 2009.

[22] Simson L. Garfinkel, David J. Malan, Karl-Alexander Dubec, Christopher C. Stevens, and Cecile Pham. 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, January 2006.

[23] Gregory Grefenstette. Implementing a content-based public-oriented audio and video news retrieval system. In *Proceedings of the international conference on Multimedia information retrieval*, MIR '10, pages 11–12. ACM, New York, NY, USA, 2010. ISBN 978-1-60558-815-5. http://doi.acm.org/10.1145/1743384.1743393.

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

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

[26] Yinghua Guo and Jill Slay Jason Beckett. Validation and verification of computer forensic software tools—searching function. *Digital Investigation*, 6:S12–S22, 2010.

[27] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52:91–98, May 2009. ISSN 0001-0782. http://doi.acm.org/10.1145/1506409.1506429.

[28] Vagelis Hristidis, Oscar Valdivia, Michail Vlachos, and Philip S. Yu. Information discovery across multiple streams. *Inf. Sci.*, 179:3268–3285, September 2009. ISSN 0020-0255. http://portal.acm.org/citation.cfm?id=1576858.1576995.

[29] Ross Johnson. POSIX Threads (pthreads) for Win32, 2006. http://sourceware.org/pthreads-win32/.

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

[31] Gerwin Klein. Jflex - the fast scanner generator for java, 2009. http://jflex.de.

[32] Bas Kloet, Joachim Metz, Robert-Jan Mora, David Loveall, and Dennis Schreiber. libewf: Project info, 2008. http://www.uitwisselplatform.nl/projects/libewf/, Uitwisselplatform.NL.

[33] Charalampos Konstantopoulos, Basilis Mamalis, Grammati Pantziou, and Damianos Gavalas. Efficient parallel text retrieval techniques on bulk synchronous parallel (bsp)/coarse grained multicomputers (cgm). *J. Supercomput.*, 48:286–318, June 2009. ISSN 0920-8542. http://portal.acm.org/citation.cfm?id=1555972.1555999.

[34] John Lehr. Personal communication, June 3 2010.

[35] Jim Lyle. Unicode string searching—russian text, 2008. http://www.cfreds.nist.gov/utf-16-russ.html, National Institute of Standards and Technology.

[36] Nick Mikus, Kris Kendall, and Jesse Kornblum. Foremost(1), January 2006. http://foremost.sourceforge.net/foremost.html. [Online; accessed 06 March 2009].

[37] B.P. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of UNIX utilities. *Communications of the ACM*, 33, December 1990.

[38] Robert-Jan Mora. Digital forensic sampling, March 29 2010. http://blogs.sans.org/computer-forensics/2010/03/29/digital-forensic-sampling/.

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

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

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

[42] Computer Forensic Tool Testing Program. Forensic string searching tool requirements specification, January 24 2008. http://www.cftt.nist.gov/ss-req-sc-draft-v1_0.pdf, National Institute of Standards and Technology.

[43] Golden G. Richard, III and Vassil Roussev. Next-generation digital forensics. *Commun. ACM*, 49(2):76–80, 2006. ISSN 0001-0782.

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

[45] Vassil Rosusev. Personal communication, 2006. Group Discussion at DFRWS 2006.

[46] Secretary of State for the Home Department. Counter-terrorism policy and human rights: 28 days, intercept and post-charge questioning, September 2007. http://www.official-documents.gov.uk/document/cm72/7215/7215.pdf. The Government Reply To The Nineteenth Report from the Joint Committee on Human Rights Session 2006-07 HL Paper 157, HC 394.

[47] Ewald Snel. NFI defraser, 2010. http://sourceforge.net/projects/defraser/.

[48] Radhakrishnan Srinivasan. Speed web delivery with http compression, 2003. http://www.ibm.com/developerworks/web/library/wa-httpcomp/, IBM Developer Works.

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

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

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

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

[53] Michael Wei, Laura M. Grupp, Frederick E. Spada, and Steven Swanson. Reliably erasing data from flash-based solid state drives. In *9th USENIX Conference on File and Storage Technologies*, 2011.

[54] Eugene Weinstein. *Search problems for speech and audio sequences*. PhD thesis, New York, NY, USA, 2009. AAI3380255.