

bulk extractor

USER MANUAL

Quickstart Guide Included
July 8, 2014

Authored by:
Jessica R. Bradley
Simson L. Garfinkel

One Page Quickstart for Linux & Mac Users

This page provides a very brief introduction to downloading, installing and running *bulk_extractor*.

1. If you do not already have one, obtain a disk image on which to run *bulk_extractor*. Sample images can be downloaded from <http://digitalcorpora.org/corpora/disk-images>. Suggestions include `nps-2009-domexusers` and `nps-2009-ubnist1.gen3`.
2. Download the latest version of *bulk_extractor*. It can be obtained from http://digitalcorpora.org/downloads/bulk_extractor/. The file is called `bulk_extractor-x.y.z.tar.gz` where x.y.z is the latest version.
3. Un-tar and un-zip the file. In the newly created *bulk_extractor-x.y* directory, run the following commands:

```
■ ./configure
■ make
■ sudo make install
```

[Refer to **Subsubsection 3.1.1 Installing on Linux or Mac**. Note, for full functionality, some users may need to first download and install dependent library files. Instructions are outlined in the referenced section.]

4. To run *bulk_extractor* from the command line, type the following command:

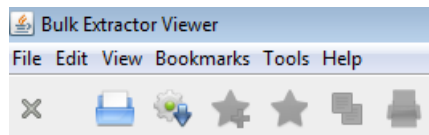
```
■ bulk_extractor -o output mydisk.raw
```

In the above command, **output** is the directory that will be created to store *bulk_extractor* results. It can not already exist. The input **mydisk.raw** is the disk image to be processed. [See **Subsection 3.2 Run *bulk_extractor* from the Command Line**]

5. To run *bulk_extractor* from the **Bulk Extractor Viewer**, navigate to the directory called `/java_gui` in the *bulk_extractor* folder and run the following command:

```
■ ./BEViewer
```

In the **Bulk Extractor Viewer**, click on the Gear/down arrow icon as depicted below.



A window will pop up and the first two input boxes allow you to select an Image File and specify an Output Feature Directory to create. Enter both of those and then select the button at the bottom of the window titled "Start *bulk_extractor*" to run *bulk_extractor*. [See **Subsection 3.3 Run *bulk_extractor* from Bulk Extractor Viewer**]

6. Whether *bulk_extractor* was run from the command line or the **Bulk Extractor Viewer** tool, after the run the resulting output files will be contained in the specified output directory. Open that directory and verify files have been created. There should be 15-25 files. Some will be empty and others will be populated with data.
7. Users can join the google email users group for more information and help with any issues encountered. Email **bulk_extractor-users+subscribe@googlegroups.com** with a blank message to join.

One Page Quickstart for Windows Users

This page provides a very brief introduction to downloading, installing and running *bulk_extractor*.

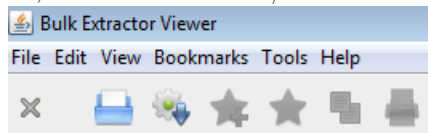
1. If you do not already have one, obtain a disk image on which to run *bulk_extractor*. Sample images can be downloaded from <http://digitalcorpora.org/corpora/disk-images>. Suggestions include `nps-2009-domexusers` and `nps-2009-ubnist1.gen3`.
2. Download the latest version of the *bulk_extractor* Windows installer. It can be obtained from http://digitalcorpora.org/downloads/bulk_extractor. The file to download is called `bulk_extractor-x.y.z-windowsinstaller.exe` where `x.y.z` is the latest version number. Run the installer file. This will automatically install *bulk_extractor* on your machine. The automatic installation includes the complete *bulk_extractor* system as well as the **Bulk Extractor Viewer** tool. [See Subsubsection 3.1.2 Installing on Windows]
3. To run *bulk_extractor* from the command line, type the following command:

```
■ bulk_extractor -o output mydisk.raw
```

In the above command, **output** is the directory that will be created to store *bulk_extractor* results. It can not already exist. The input **mydisk.raw** is the disk image to be processed. [See Subsection 3.2 Run *bulk_extractor* from the Command Line]

4. To run *bulk_extractor* from the **Bulk Extractor Viewer**, run the program **Bulk Extractor X.Y** from the Start Menu.

In the **Bulk Extractor Viewer**, click on the Gear/down arrow icon as depicted



below.

A window will pop up and the first two input boxes allow you to select an Image File and specify an Output Feature Directory to create. Enter both of those and then select the button at the bottom of the window titled "Start *bulk_extractor*" to run *bulk_extractor*. [See Subsection 3.3 Run *bulk_extractor* from Bulk Extractor Viewer]

5. Whether *bulk_extractor* was run from the command line or the **Bulk Extractor Viewer** tool, after the run the resulting output files will be contained in the specified output directory. Open that directory and verify files have been created. There should be 15-25 files. Some will be empty and others will be populated with data.
6. Users can join the google email users group for more information and help with any issues encountered. Email `bulk_extractor-users+subscribe@googlegroups.com` with a blank message to join.

Contents

1	Introduction	1
1.1	Overview of <i>bulk_extractor</i>	1
1.1.1	A <i>bulk_extractor</i> Success Story	2
1.2	Purpose of this Manual	3
1.3	Conventions Used in this Manual	3
2	How <i>bulk_extractor</i> Works	3
3	Running <i>bulk_extractor</i>	6
3.1	Installation Guide	6
3.1.1	Installing on Linux or Mac	6
3.1.2	Installing on Windows	8
3.2	Run <i>bulk_extractor</i> from the Command Line	10
3.3	Run <i>bulk_extractor</i> from Bulk Extractor Viewer	12
3.4	Run <i>bulk_extractor</i> from Bulk Extractor Viewer	12
4	Processing Data	17
4.1	Types of Input Data	17
4.2	Scanners	21
4.3	Carving	23
4.4	Suppressing False Positives	24
4.5	Using an Alert List	26
4.6	The Importance of Compressed Data Processing	26
5	Use Cases for <i>bulk_extractor</i>	27
5.1	Malware Investigations	27
5.2	Cyber Investigations	28
5.3	Identity Investigations	29
5.4	Password Cracking	32
5.5	Analyzing Imagery Information	32
5.6	Using <i>bulk_extractor</i> in a Highly Specialized Environment	32
6	Tuning <i>bulk_extractor</i>	33
7	Post Processing Capabilities	33
7.1	<code>bulk_diff.py</code> : Difference Between Runs	34
7.2	<code>identify_filenames.py</code> : Identify File Origin of Features	34
8	Worked Examples	34
8.1	Encoding	35
9	2009-M57 Patents Scenario	35
9.1	Run <i>bulk_extractor</i> with the Data	35
9.2	Digital Media Triage	38
9.3	Analyzing Imagery	43
9.4	Password Cracking	44
9.5	Post Processing	46

10 NPS DOMEX Users Image	47
10.1 Malware Investigations	49
10.2 Cyber Investigations	51
11 Troubleshooting	53
12 Related Reading	54
Appendices	56
A Output of <i>bulk_extractor</i> Help Command	56

1 Introduction

1.1 Overview of *bulk_extractor*

bulk_extractor is a program that extracts features such as email addresses, credit card numbers, URLs, and other types of information from digital evidence files. It is a useful forensic investigation tool for many tasks such as malware and intrusion investigations, identity investigations and cyber investigations, as well as analyzing imagery and password cracking. The program provides several unusual capabilities including:

- It finds email addresses, URLs and credit card numbers that other tools miss because it can process compressed data (like ZIP, PDF and GZIP files) and incomplete or partially corrupted data. It can carve JPEGs, office documents and other kinds of files out of fragments of compressed data. It will detect and carve encrypted RAR files.
- It builds word lists based on all of the words found within the data, even those in compressed files that are in unallocated space. Those word lists can be useful for password cracking.
- It is multi-threaded; running *bulk_extractor* on a computer with twice the number of cores typically makes it complete a run in half the time.
- It creates histograms showing the most common email addresses, URLs, domains, search terms and other kinds of information on the drive.

bulk_extractor operates on disk images, files or a directory of files and extracts useful information without parsing the file system or file system structures. The input is split into pages and processed by one or more scanners. The results are stored in feature files that can be easily inspected, parsed, or processed with other automated tools. *bulk_extractor* also creates histograms of features that it finds. This is useful because features such as email addresses and internet search terms that are more common tend to be important.

In addition to the capabilities described above, *bulk_extractor* also includes:

- A graphical user interface, **Bulk Extractor Viewer**, for browsing features stored in feature files and for launching *bulk_extractor* scans
- A small number of python programs for performing additional analysis on feature files

bulk_extractor 1.4 detects and optimistically decompresses data in ZIP, GZIP, RAR, and Microsoft's Hibernation files. This has proven useful, for example, in recovering email addresses from fragments of compressed files found in unallocated space.

bulk_extractor contains a simple but effective mechanism for protecting against decompression bombs. It also has capabilities specifically designed for Windows and malware analysis including decoders for the Windows PE, Linux ELF, VCARD, Base16, Base64 and Windows directory formats.

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.

Threading is accomplished through the use of an analysis thread pool. After the features have been extracted, *bulk_extractor* builds a histogram of email addresses, Google search terms, and other extracted features. Stop lists can also be used to remove features not relevant to a case.

bulk_extractor is distinguished from other forensic tools by its speed and thoroughness. Because it ignores file system structure, *bulk_extractor* can process different parts of the disk in parallel. This means that an 8-core machine will process a disk image roughly 8 times faster than a 1-core machine. *bulk_extractor* is also thorough. It automatically detects, decompresses, and recursively re-processes data that has been compressed with a variety of algorithms. Our testing has shown there is a significant amount of compressed data in the unallocated regions of file systems missed by most forensics tools that are commonly in use today[?]. Another advantage of ignoring file systems is that *bulk_extractor* can be used to process any kind of digital media. The program has been used to process hard drives, SSDs, optical media, camera cards, cell phones, network packet dumps, and other kinds of digital information.

Between 2005 and 2008, the *bulk_extractor* team interviewed law enforcement regarding their use of forensic tools. Law enforcement officers wanted a highly automated tool for finding email addresses and credit card numbers (including track 2 information), phone numbers, GPS coordinates and EXIF information from JPEGs, search terms (extracted from URLs), and all words that were present on the disk (for password cracking). The tool needed to run on Windows, Linux and Mac-based systems with no user interaction. It also had to operate on raw disk images, split-raw volumes and E01 files. The tool needed to run at the maximum I/O speed of the physical drive and never crash. Through these interviews, the initial requirements for the *bulk_extractor* system were developed. Over the past five years, we have worked to create the tool that those officers desired.

1.1.1 A *bulk_extractor* Success Story

One early *bulk_extractor* success story comes from the City of San Luis Obispo Police Department in the Spring of 2010. The District Attorney filed charges against two individuals for credit card fraud and possession of materials to commit credit card fraud. The defendants were arrested with a computer. Defense attorneys were expected to argue that the defendants were unsophisticated and lacked knowledge to commit the crime. The examiner was given a 250 GB drive the day before the preliminary hearing; typically it would take several days to conduct a proper forensic investigation of that much data.

bulk_extractor found actionable evidence in only two and a half hours including the following information:

- There were over 10,000 credit card numbers on the hard drive (illegal materials). Over 1000 of the credit card numbers were unique.
- The most common email address belonged to the primary defendant (evidence of possession).
- The most commonly occurring internet search engine queries concerned credit card fraud and bank identification numbers (evidence of intent).

- The most commonly visited websites were in a foreign country whose primary language is spoken by the defendant (evidence of flight risk).

Armed with this data, the defendants were held without bail.

As *bulk_extractor* has been deployed and used in different applications, it has evolved to meet additional requirements. This manual describes use cases for the *bulk_extractor* system and demonstrates how users can take full advantage of all of its capabilities.

1.2 Purpose of this Manual

This User Manual is intended to be useful to new, intermediate and experienced users of *bulk_extractor*. It provides an in-depth review of the functionality included in *bulk_extractor* and shows how to access and utilize features through both command line operation and the **Bulk Extractor Viewer**. This manual includes working examples with links to the input data (disk images) used, giving users the opportunity to work through the examples and utilize all aspects of the system.

1.3 Conventions Used in this Manual

This manual uses standard formatting conventions to highlight file names, directory names and example commands. The conventions for those specific types are described in this section.

Names of programs including the post-processing tools native to *bulk_extractor* and third-party tools are shown in **bold**, as in **tcpflow**.

File names are displayed in a fixed width font. They will appear as `filename.txt` within the text throughout the manual.

Directory names are displayed in italics. They appear as *directoryname/* within the text. The only exception is for directory names that are part of an example command. Directory names referenced in example commands appear in the example command format.

Scanner names are denoted with bold, italicized text. They are always specified in lower-case, because that is how they are referred in the options and usage information for *bulk_extractor*. Names will appear as ***scannername***.

This manual contains example commands that should be typed in by the user. A command entered at the terminal is shown like this:

■ **command**

The first character on the line is the terminal prompt, and should not be typed. The black square is used as the standard prompt in this manual, although the prompt shown on a users screen will vary according to the system they are using.

2 How *bulk_extractor* Works

bulk_extractor finds email addresses, URLs, and CCNs that other tools miss. This is due in part to the fact that *bulk_extractor* optimistically decompresses and re-analyzes

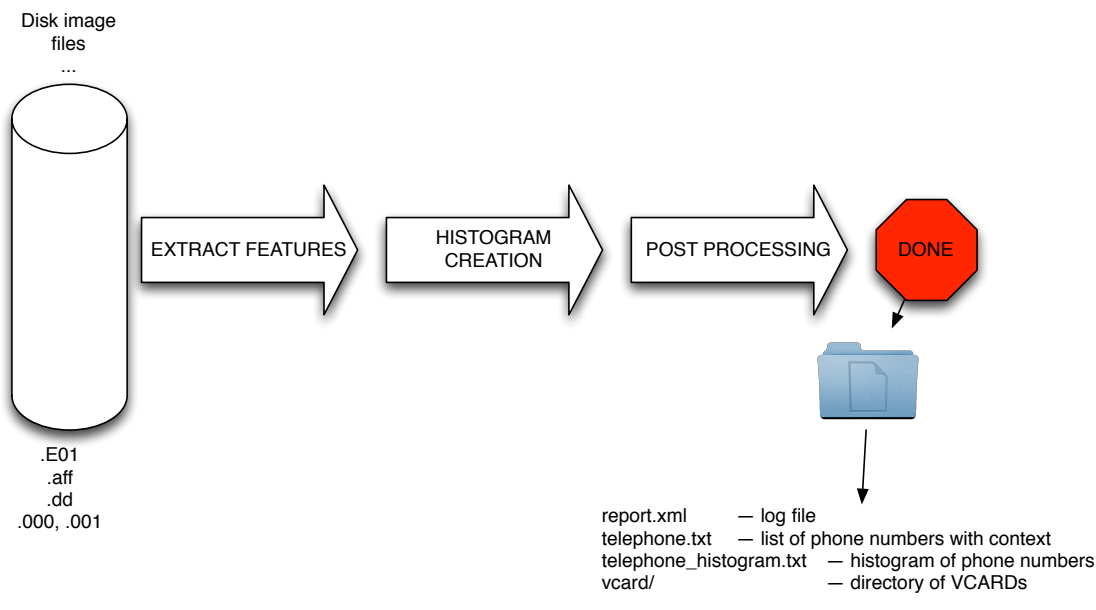


Figure 1: Three Phases of *bulk_extractor* Operation

all data (e.g. zip fragments, gzip browser cache runs). The decompression operates on incomplete and corrupted data until decompression fails. *bulk_extractor* can also build word lists for password cracking

There are three phases of operation in *bulk_extractor*: feature extraction, histogram creation, post processing as shown in Figure 1. The output feature files contain extracted data designed for easy processing by third party programs or use in spreadsheet tools. The *bulk_extractor* histogram system automatically summarizes features.

Features files are written using the feature recording system. As features are discovered, they are sent to the feature recorder and recorded in the appropriate file. Multiple scanners might write to the same feature file. For example, the *exif* scanner searches the file formats used by digital cameras and finds GPS coordinates in images. Those findings are written to the output file *gps.txt* by the *gps* feature recorder. A separate scanner, the *gps* scanner, searches Garmin Trackpoint data and also finds GPS coordinates and writes them to *gps.txt*. It is worth noting that some scanners also find more than one type of feature and write to several feature files. For example, the *email* scanner looks for email addresses, domains, URLs and RFC822 headers and writes them to *email.txt*, *domain.txt*, *url.txt*, *rfc822.txt* and *ether.txt* respectively.

A feature file contains rows of features. Each row is typically comprised of an offset, a feature, and the feature in evidence context although scanners are free to store whatever information they wish. A few lines of an email feature file might look like the following:

OFFSET	FEATURE	FEATURE IN EVIDENCE CONTEXT
48198832	domexuser2@gmail.com	__<name>domexuser2@gmail.com/Home
48200361	domexuser2@live.com	__<name>domexuser2@live.com</name>
48413823	siege@preoccupied.net	'Brien <siege@preoccupied.net>_1

The types of features displayed in the feature file will vary depending on what type of

feature is being stored. However, all feature files use the same format with each row corresponding to one found instance of a feature and three columns describing the related data (offset, feature, and feature in evidence context).

Histograms are a powerful tool for understanding certain kinds of evidence. A histogram of emails allows us to rapidly determine the drive's primary user, the user's organization, primary correspondents and other email addresses. The feature recording system automatically makes histograms as data are processed. When the scanner writes to the feature recording system, the relevant histograms are automatically updated.

A histogram file will, in general, look like the following file excerpt:

```
n=875 mozilla@kewis.ch (utf16=3)
n=651 charlie@m57.biz (utf16=120)
n=605 ajbanck@planet.nl
...
n=288 mattwillis@gmail.com
n=281 garths@oeone.com
n=226 michael.buettner@sun.com (utf16=2)
n=225 bugzilla@babylonsounds.com
n=218 berend.cornelius@sun.com
n=210 ips@mail.ips.es
n=201 mschroeder@mozilla.x-home.org
n=186 pat@m57.biz (utf16=1)
```

Each line shows a feature and the number of times that feature was found by *bulk_extractor* (the histogram indicates how many times the item was found coded as UTF-16). Features are stored in the file in order of occurrence with most frequent features appearing at the top of the file and least frequent displayed at the bottom.

bulk_extractor has multiple scanners that extract features. Each scanner runs in an arbitrary order. Scanners can be enabled or disabled which can be useful for debugging and speed optimization. Some scanners are recursive and actually expand the data they are exploring, thereby creating more data that *bulk_extractor* can analyze. These blocks are called sbufs. The "s" stands for the word safe. All access to data in the sbuf is bounds-checked, so buffer overflow events are very unlikely. The sbuf data structure is one of the reasons that *bulk_extractor* is so crash resistant. Recursion is used for, among other things, decompressing ZLIB and Windows HIBERFILE, extracting text from PDFs and handling compressed browser cache data.

The recursion process requires a new way to describe offsets. To do this, *bulk_extractor* introduces the concept of the "forensic path." The forensic path is a description of the origination of a piece of data. It might come from, for example, a flat file, a data stream, or a decompression of some type of data. Consider an HTTP stream that contains a GZIP-compressed email as shown in Figure 2. A series of scanners will first find the ZLIB compressed regions in the HTTP stream that contain the email, decompress them, and then find the features in that email which may include email addresses, names and phone numbers. Using this method, *bulk_extractor* can find email addresses in compressed data. The forensic path for the email addresses found indicate that it originated in an email, that was GZIP compressed and found in an HTTP stream. The forensic path of the email addresses features found might be represented as follows:

```
11052168704-GZIP-3437 live.com eMn='domexuser@live.com';var srf_sDispM
11052168704-GZIP-3475 live.com pMn='domexuser@live.com';var srf_sDreCk
```

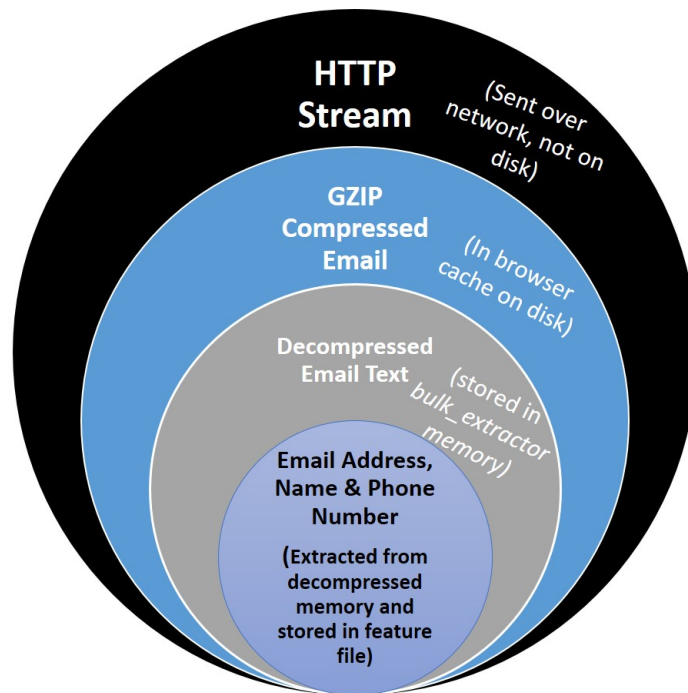


Figure 2: Forensic path of features found in email lead back to HTTP Stream

```
11052168704-GZIP-3512 live.com eCk='domexuser@live.com';var srf_sFT='<
```

The full functionality of *bulk_extractor* is provided both through command line operation and the GUI tool, **Bulk Extractor Viewer**. Both modes of operation work for Linux, Mac and Windows. The following section describes how to download, install and run *bulk_extractor* using either the command line or the **Bulk Extractor Viewer**.

3 Running *bulk_extractor*

bulk_extractor is a command line tool with an accompanying graphical user interface tool, **Bulk Extractor Viewer**. All of the command line functionality of *bulk_extractor* is also available in the **Bulk Extractor Viewer**. Users can access the functionality in whichever way they prefer. In this manual we review the *bulk_extractor* user options in both formats.

bulk_extractor can be run on a Linux, MacOS or Windows system. The fastest way to run *bulk_extractor* is on a Linux system. Running on Windows provides the same results, but the run will typically take 40

3.1 Installation Guide

Installation instructions vary for Linux/Mac users and Windows users. The following sections explain how to install *bulk_extractor*.

3.1.1 Installing on Linux or Mac

Before compiling *bulk_extractor* for your platform, you may need to install other packages on your system which *bulk_extractor* requires to compile cleanly and with a full

set of capabilities.

Dependencies for Linux Fedora

This command should add the appropriate packages:

- `sudo yum update`
- `sudo yum groupinstall development-tools`
- `sudo yum install flex`

Dependencies for Linux Debian Testing (wheezy) or Ubuntu 13.0

The following command should add the appropriate libraries:

- `sudo apt-get -y install gcc g++ flex libewf-dev`

Dependencies for Mac Systems

Mac users must first install Apple's Xcode development system. Other components should be downloaded using the MacPorts system. If you do not have MacPorts, go to the App store and download and install it. It is free. Once it is installed, try:

- `sudo port install flex autoconf automake libewf-devel`

Mac users should note that libewf-devel may not be available in ports. If it is not, download and un-tar the libewf source, cd into the directory and run:

- `./configure`
- `make`
- `sudo make install`

Download and Install *bulk_extractor*

Next, download the latest version of *bulk_extractor*. The software can be downloaded from http://digitalcorpora.org/downloads/bulk_extractor/. The file to download will be called `bulk_extractor-x.y.z.tar.gz` where x.y.z is the latest version. As of publication of this manual, the latest version of *bulk_extractor* is 1.4.0.

After downloading the file, un-tar it. Then, in the newly created *bulk_extractor-x.y.z* directory, run the following commands to install *bulk_extractor* in `/usr/local/bin` (by default):

- `./configure`
- `make`
- `sudo make install`

With these instructions, the following directory will not be installed:

- *plugins/* - This is for C/C++ developers only. You can develop your own *bulk_extractor* plugins which will then be run at run-time with the *bulk_extractor* executable. Refer to the ***bulk_extractor* Programmers Manual for Developing Scanner Plug-ins** [?] for more information.

Instructions on running *bulk_extractor* from the command line can be found in **Subsection 3.2**.

The **Bulk Extractor Viewer** tool is installed as part of the above installation process. Specific instructions on running it can be found in **Subsection 3.3**.

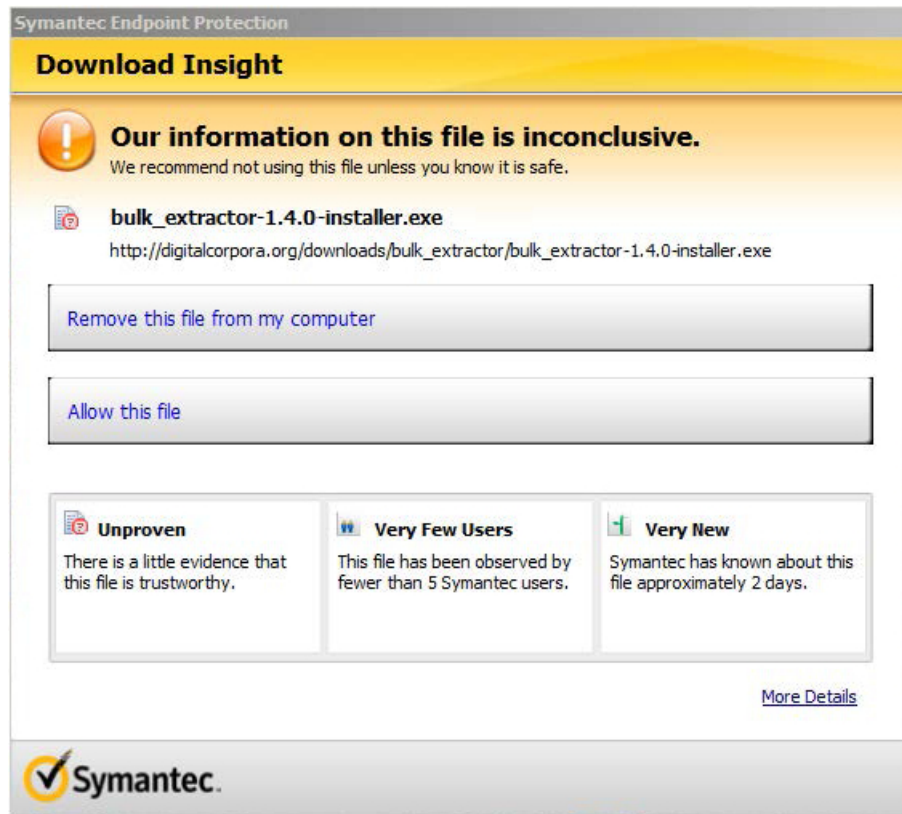


Figure 3: Anti-virus software, such as Symantec, often tries to block download of the installer file

3.1.2 Installing on Windows

Windows users should download the Windows Installer for *bulk_extractor*. The file to download is located at http://digitalcorpora.org/downloads/bulk_extractor/executables/ and is called *bulk_extractor-x.y.z-windowsinstaller.exe* where x.y.z is the latest version number (1.4.0 - as of publication of this manual).

Next, run the *bulk_extractor-x.y.z-windowsinstaller.exe* file. This will automatically install *bulk_extractor* on your machine. Because this file is not used by many Windows users, some anti-virus systems will try to manually delete it on download or block the download as shown in Figure 3. Be aware that you may have to work around your anti-virus system. Additionally, some Windows versions will try to prevent you from running it. Figure 4 shows the message Windows 8 displays when trying to run the installer. To run anyway, click on "More info" and then select "Run Anyway."

When the installer file is executed, the installation will begin and show a dialog like the one shown in Figure 5. Users should select the default configuration, which will be the 64-bit configuration for 64-bit Windows systems, or the 32-bit configuration for 32-bit Windows systems. Click on "Install" and the installer will install *bulk_extractor* on your system and then notify you when it is complete.

The automatic installation includes the **Bulk Extractor Viewer** tool as well as the complete *bulk_extractor* system that can be run from the command line. Java 6 or above

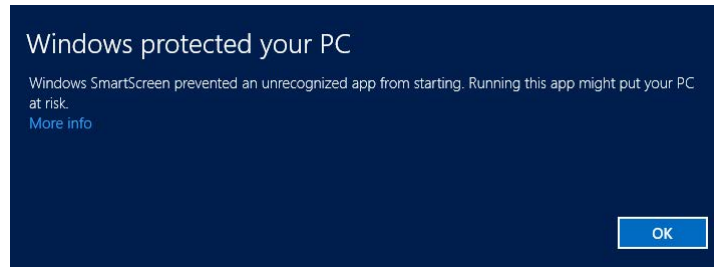


Figure 4: Windows 8 warning when trying to run the installer

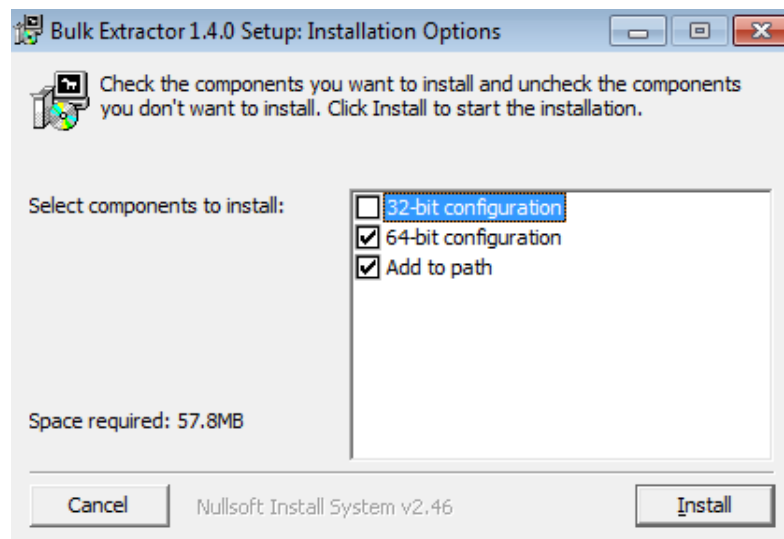


Figure 5: Dialog appears when the user executes the Windows Installer

must be installed on the machine for the **Bulk Extractor Viewer** to run. Instructions on running *bulk_extractor* from the command line can be found in **Subsection 3.2**. Instructions on running it from the **Bulk Extractor Viewer** are located in **Subsection 3.3**.

3.2 Run *bulk_extractor* from the Command Line

The two main parameters required to run *bulk_extractor* are an output directory and a disk image. The output directory must be a directory that does not already exist. The disk image can be either a file such as a disk image or a directory of individual files. *bulk_extractor* cannot process a directory of disk images.

In the following instructions, *output* is the name of the directory that will be created to store the *bulk_extractor* output. The file *mydisk.raw* is the name of the disk image that will be extracted by *bulk_extractor*.

To run *bulk_extractor* from the command line on any machine, type the following command:

```
■ bulk_extractor -o output mydisk.raw
```

The above command on any of the supported operating systems assumes that the disk image *mydisk.raw* is located in the directory where the command is being executed. However, you can point *bulk_extractor* to a disk image found elsewhere on your machine by explicitly entering the path to that image.

The following text shows the output that is produced when *bulk_extractor* is run on the file *nps-2010-emails.E01*. The information printed indicates the version number, input file, output directory and disk size. The screen is updated as *bulk_extractor* runs with status information. *bulk_extractor* then prints performance information and the number of features found when the run is complete.

```
C:\>bulk_extractor -o bulk_extractor\Output\nps-2010-emails bulk_extractor\In  
putData\nps-2010-emails.E01
```

```
bulk_extractor version: 1.4.0  
Input file: bulk_extractor\InputData\nps-2010-emails.E01  
Output directory: bulk_extractor\Output\nps-2010-emails  
Disk Size: 10485760  
Threads: 4  
All data are read; waiting for threads to finish...  
Time elapsed waiting for 1 thread to finish:  
    (timeout in 60 min .)  
Time elapsed waiting for 1 thread to finish:  
    6 sec (timeout in 59 min 54 sec.)  
Thread 0: Processing 0
```

```
All Threads Finished!
```

```
Producer time spent waiting: 0 sec.
```

```
Average consumer time spent waiting: 8.32332 sec.
```

```
Phase 2. Shutting down scanners
```

```
Phase 3. Creating Histograms
```

```
ccn histogram...    ccn_track2 histogram...    domain histogram...  
email histogram...    ether histogram...    find histogram...  
ip histogram...    lightgrep histogram...    tcp histogram...  
telephone histogram...    url histogram...    url microsoft-live...
```

```

url services... url facebook-address... url facebook-id...
url searches...Elapsed time: 11.1603 sec.
Overall performance: 0.939557 MBytes/sec
Total email features found: 67

```

Note that *bulk_extractor* has automatically selected to use 4 threads; this is because the program was run on a computer with 4 cores. In general, *bulk_extractor* automatically determines the correct number of cores to use. It is not necessary to set the number of threads to use.

After running *bulk_extractor*, examine the output directory specified by name in the run command. There should now be a number of generated output files in that directory. There are several categories of output created for each *bulk_extractor* run. First, there are feature files grouped by category, which contain the features found and include the path, feature and context. Second, there are histogram files that allow users to quickly see the features grouped by the frequency in which they occur. Certain kinds of files, such as JPEGs and KML files, may be carved into directories. Finally, *bulk_extractor* creates a file **report.xml**, in DFXML format, that captures the provenance of the run. After *bulk_extractor* has been run, all of these files will be found in the output directory specified by the user.

The text below shows the results of running the command **ls -s** within the output directory from the *bulk_extractor* run on the disk image **nps-2010-emails.E01**. The numbers next to the file names indicate the file size and show that several of the files, including **email.txt** and **domain.txt**, were populated with features during the run.

```

C:\bulk_extractor\Output\nps-2010-emails>ls -s
total 303
 0 aes_keys.txt          0 kml.txt
 0 alerts.txt           0 lightgrep.txt
 0 ccn.txt              0 lightgrep_histogram.txt
 0 ccn_histogram.txt    0 rar.txt
 0 ccn_track2.txt       8 report.xml
 0 ccn_track2_histogram.txt 0 rfc822.txt
64 domain.txt          0 tcp.txt
 1 domain_histogram.txt 0 tcp_histogram.txt
 0 elf.txt              0 telephone.txt
16 email.txt           0 telephone_histogram.txt
 4 email_histogram.txt 96 url.txt
 0 ether.txt            0 url_facebook-address.txt
 0 ether_histogram.txt  0 url_facebook-id.txt
 1 exif.txt             4 url_histogram.txt
 0 find.txt             0 url_microsoft-live.txt
 0 find_histogram.txt   0 url_searches.txt
 0 gps.txt              1 url_services.txt
 0 hex.txt              0 vcard.txt
 0 ip.txt               12 windirs.txt
 0 ip_histogram.txt     0 winpe.txt
 0 jpeg                 0 winprefetch.txt
 8 jpeg.txt            88 zip.txt
 0 json.txt

```

There are numerous feature files produced by *bulk_extractor* for each run. A feature file is a tab-delimited file that show a feature on each row. Each row includes a path, a feature and the context. The files are in UTF-8 format.

Any of the feature files created by *bulk_extractor* may have an accompanying ***_stopped.txt**

file found in the output directory. This file will show all stopped entries of that type that have been found so that users can examine those files to make sure nothing critical has been hidden. A stopped features is a feature that appears in a stop list. The stop list is a list of features that are not of concern for a particular investigation. For example, users may input a stop list file to *bulk_extractor* that contains numerous email addresses that should be ignored and not marked as a found feature. Rather than throwing away those results when they are found, *bulk_extractor* will create a file, named **email_stopped.txt** that shows all email addresses from the stop list that were found during the run. The stopped email addresses will not show up in the **email.txt** file. More information on creating and using stop lists can be found in **Subsection 4.4**.

While the above commands are all that is required for basic operation, there are numerous usage options that allow the user to affect input and output, tuning, path processing mode, debugging, and control of scanners. All of those options are described when *bulk_extractor* is run with the -h (help) option. It is important to note that the overwhelming tendency of users is to use many of these options; **however**, that is not generally recommended. Most of the time, the best way to run *bulk_extractor* is with no options specified other than -o to specify the output directory. For best performance and results users should avoid adding them in general. Only advanced users in specific cases should use these options.

Running *bulk_extractor* with only the -h option specified produces the output shown in **Appendix A**. To run any optional usage options, they should be inserted before the input and output options are specified. Specifically, the order should look like the following:

```
■ bulk_extractor [Usage Options] -o output mydisk.raw
```

The specific order in which multiple usage options are specified matters. Some of the options are discussed within the following sections for specific use cases, other options are for programmer or experimental use. In general, avoid using the options unless indicated for a specific purpose.

3.3 Run *bulk_extractor* from Bulk Extractor Viewer

On a Linux or Mac system, go to the directory where the **Bulk Extractor Viewer** is installed or specify the full path name to the jar file. It will be in the location where the *bulk_extractor* code was installed and in the sub-directory labeled *java_gui*. From that directory, run the following command to start the **Bulk Extractor Viewer**:

```
■ ./BEViewer
```

3.4 Run *bulk_extractor* from Bulk Extractor Viewer

Windows users should go to the Start menu and choose Programs->Bulk_Extractor x.y.z->BE Viewer with Bulk_extractor x.y.z (64-bit). If the 64-bit version can not be run on your machine, you can choose the 32-bit version. The Troubleshooting section describes some limits users of the 32-bit version might encounter.

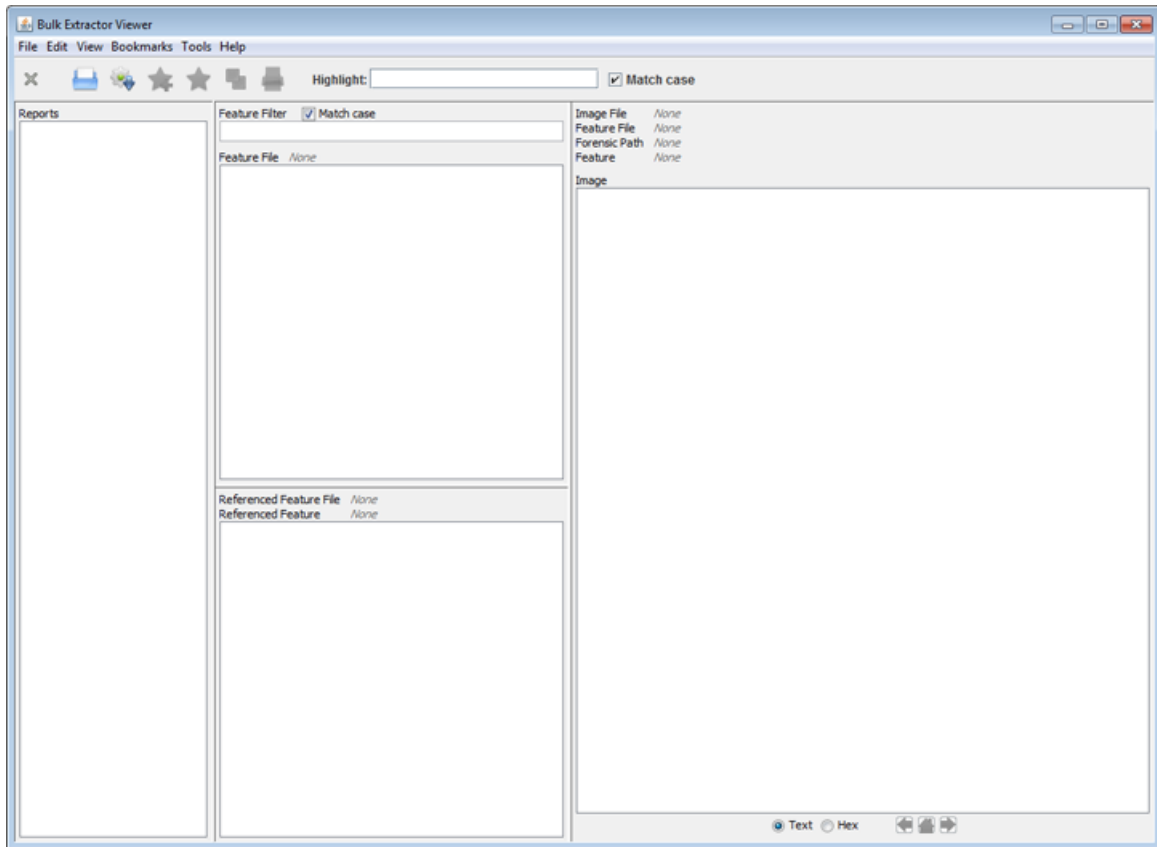


Figure 6: What **Bulk Extractor Viewer** looks like when it is started

When the **Bulk Extractor Viewer** starts up, it will look like Figure 6. The look and feel may vary slightly according to the specific operating system but all options should appear similar. To run *bulk_extractor* from the viewer, click on the icon that looks like a gear with a down arrow. It is next to the Print icon below the Tools menu. Clicking on this icon will bring up the “Run *bulk_extractor*” Window as shown in Figure 7.

Next, in the “Run *bulk_extractor*” window select the Image File and Output Feature Directory to run *bulk_extractor*. Figure 8 shows an example where the user has selected the file *nps-2010-emails.E01* as input and is going to create a directory called *nps-2010-charlie-output* in the parent directory *C:\bulk_extractor\Output*. Note that figures may vary slightly in future versions of *bulk_extractor* but the major functionality will remain the same.

After selecting the input and output directories, click on the button at the bottom of the “Run *bulk_extractor*” window labeled “Start *bulk_extractor*.” This will bring up the window shown in Figure 9 that updates as *bulk_extractor* is running, providing status information during the run and after the run is complete.

When the run is complete, a dialog will pop-up indicating the results are ready to be viewed. Figure 10 shows this dialog. Click the “Ok” button which will return you to the main **Bulk Extractor Viewer** window to view the results of the run. The “Reports” window on the left will now show the newly created report. In this example, the report is called “nps-2010-emails-output.” Clicking once on this report name will expand the

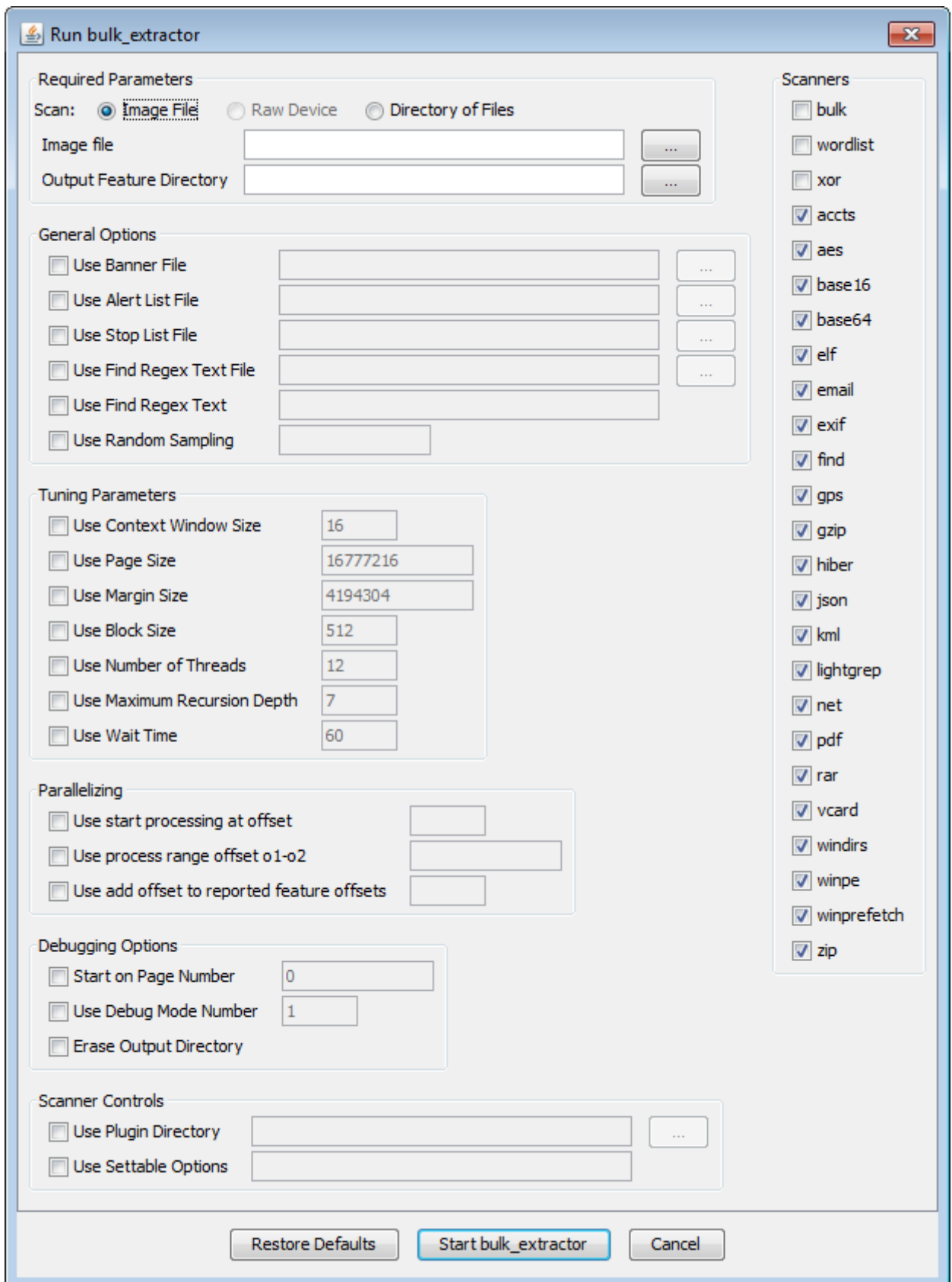


Figure 7: Clicking on the gear icon brings up this “Run bulk_extractor” Window

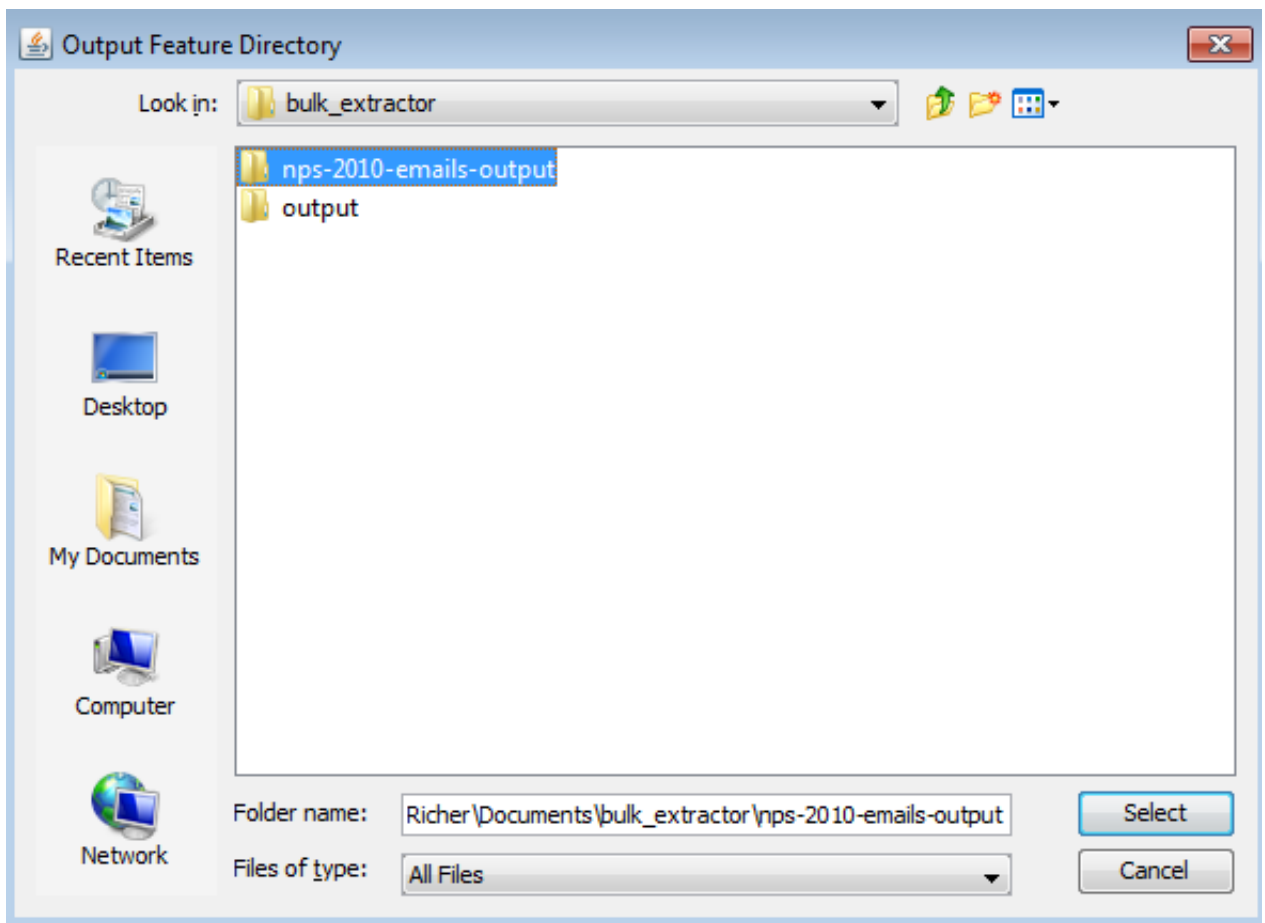


Figure 8: After selecting an Image File for input, the user must select an output directory to create

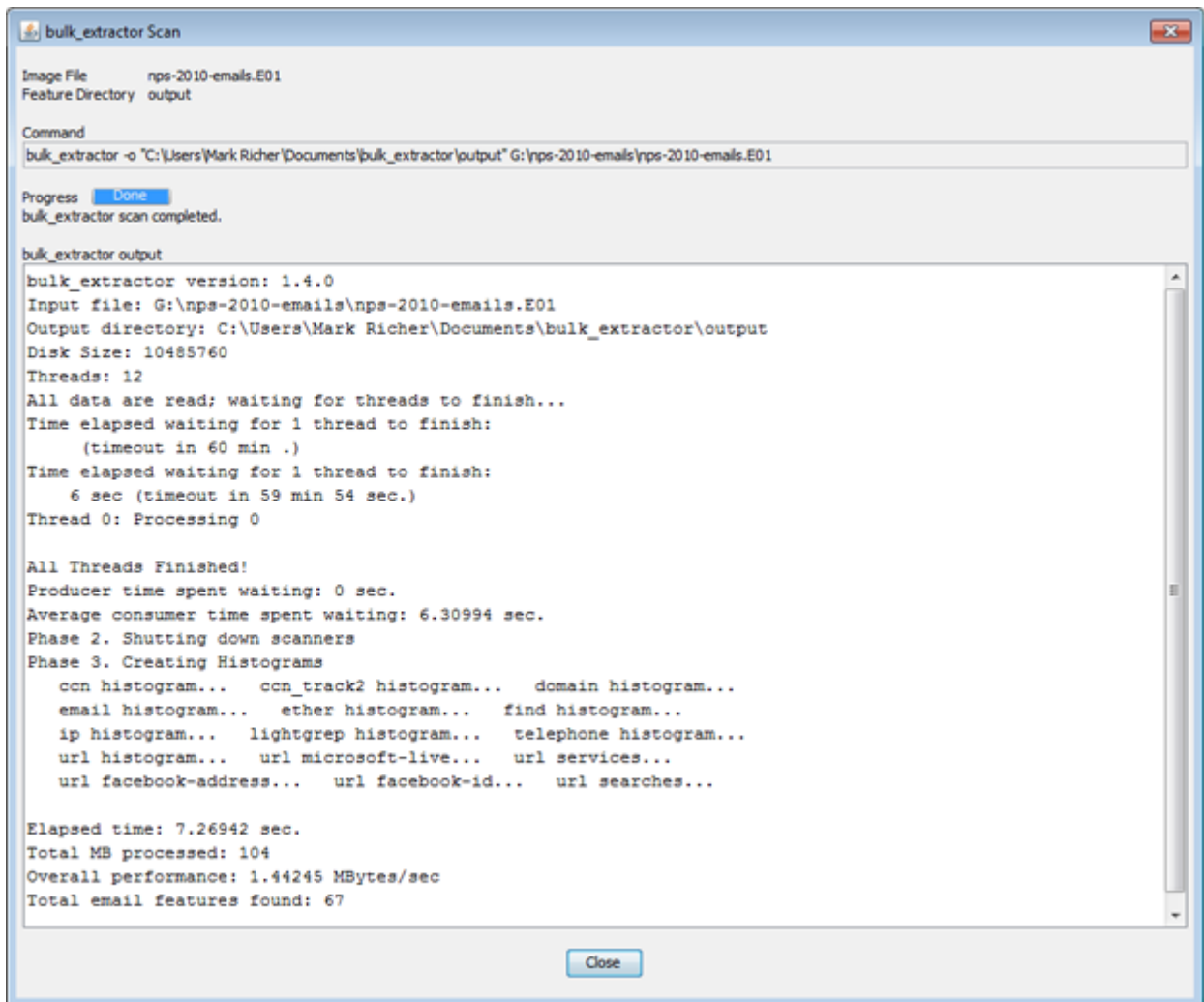


Figure 9: Status window that shows what happens as *bulk_extractor* runs and indicates when *bulk_extractor* is complete

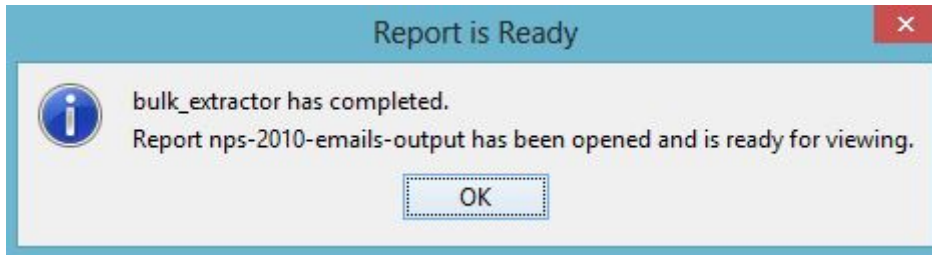


Figure 10: Dialog indicating the run of *bulk_extractor* is complete and results are ready to be viewed

report and show all of the files that have been created as shown in Figure 11.

Clicking on one of the files will bring that file up in the “Feature File” window in the middle of the screen. In the example, the user clicked on `email.txt` to view the email feature file. Clicking on one of the features, in this case `rtf_text@textedit.com`, shows the feature in context within the feature file on the right hand side of the window as shown in Figure 12.

The user can also view histogram files in the **Bulk Extractor Viewer**. Clicking on the file, `email_histogram.txt` in the Reports window on the left hand side will bring up the contents of the histogram file in the middle window. It will also display the referenced feature file in the window below the histogram file. In this case, the referenced feature file is `email.txt`. Clicking on a feature in the histogram, in this example `rtf_text@textedit.com`, will display the feature in context as found within the feature file on the right hand side of the screen as shown in Figure 13.

4 Processing Data

4.1 Types of Input Data

The *bulk_extractor* system can handle multiple image formats including E01, raw, split raw and individual disk files as well as raw devices or files. It can also operate on memory and packet captures, although packet captures will be more completely extracted if you pre-process them with **tcpflow**.

The scanners all serve different functions and look for different types of information. Often, a feature will be stored in a format not easily accessible and will require multiple scanners to extract the feature data. For example, some PDF files contain text data but the PDF format is not directly searchable by the scanner that finds email addresses or the scanner that looks for keywords. *bulk_extractor* resolves this by having the two scanners work together. The *pdf* scanner will first extract all of the text from the PDF and then the other scanners will look at the extracted text for features. This is important to remember when turning scanners off and on, as scanners work together to retrieve the features from the disk image. The types of information examined, extracted or carved by the existing *bulk_extractor* scanners are as described in Table 1, along with the scanners that process them and the specific sections where they are referenced in this manual.

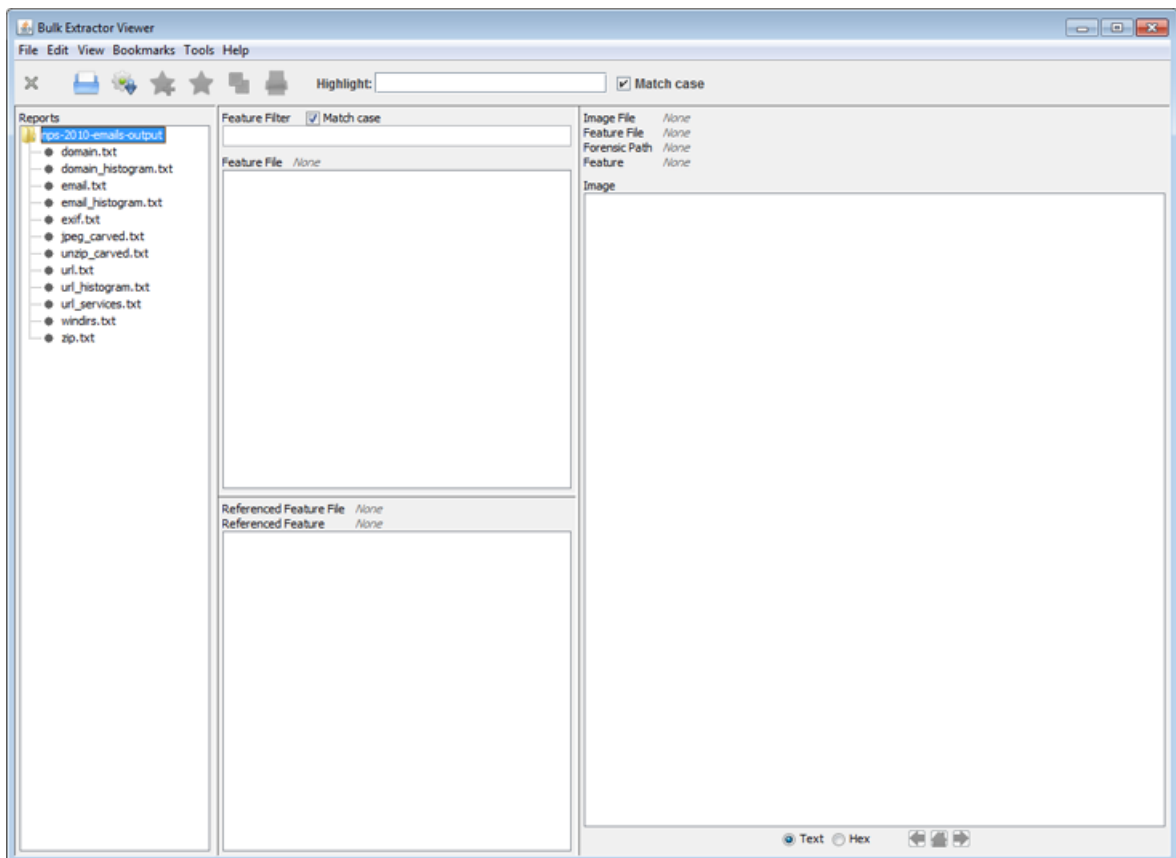


Figure 11: Reports window shows the newly created report and all of the files created in that report

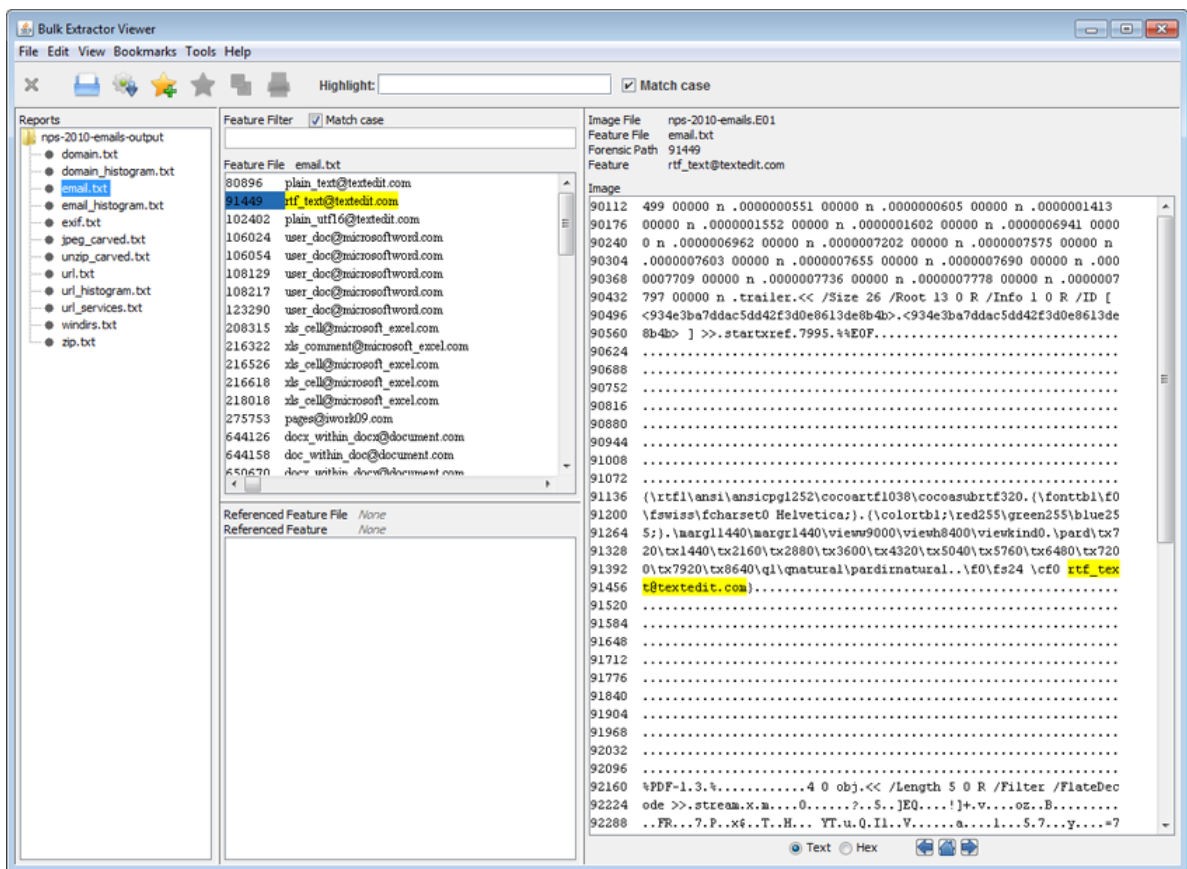


Figure 12: While viewing the feature file, the user can select a feature to view with its full context in the feature file as shown in the right hand side of the window

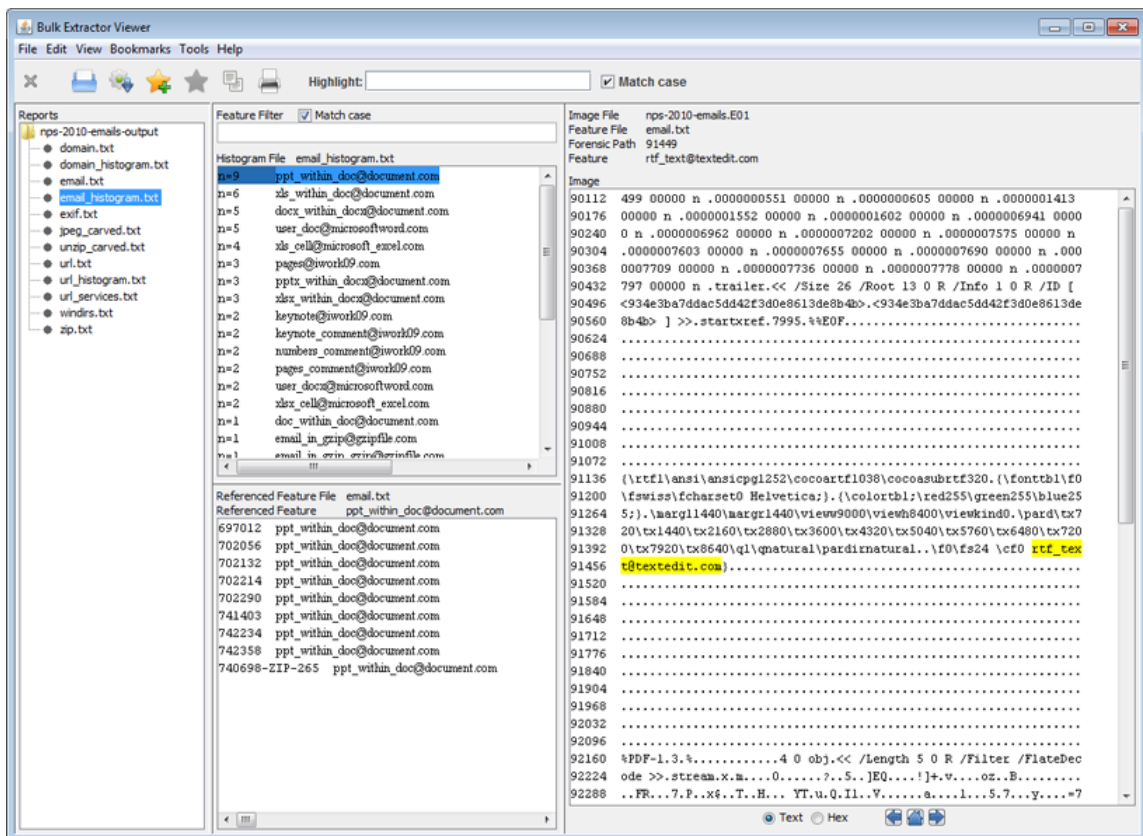


Figure 13: User can view histograms of features, referenced feature files and specific features in context

4.2 Scanners

There are multiple scanners deployed with the *bulk_extractor* system. For a detailed list of the scanners installed with your version of *bulk_extractor*, run the following command:

```
■ bulk_extractor -H
```

This command will show all of the scanners installed with additional information included about each scanner. Specifically, there is a description for each scanner, a list of the features it finds and any relevant flags. A sample of the output is below:

```
Scanner Name: accts
flags: NONE
Scanner Interface version: 3
Author: Simson L. Garfinkel
Description: scans for CCNs, track 2, and phone #s
Scanner Version: 1.0
Feature Names: alerts ccn ccn_track2 telephone
```

```
Scanner Name: base16
flags: SCANNER_RECURSE
Scanner Interface version: 3
Author: Simson L. Garfinkel
Description: Base16 (hex) scanner
Scanner Version: 1.0
Feature Names: hex
```

```
...
```

```
Scanner Name: wordlist
flags: SCANNER_DISABLED
Scanner Interface version: 3
Author:
Description:
Scanner Version:
Feature Names: wordlist
```

This output shows that the *accts* scanner looks for credit card numbers, credit card track 2 information and phone numbers and finds the feature names alerts, ccn, ccn_track2 and telephone. This means it writes to the feature files *alerts.txt*, *ccn.txt*, *ccn_track2.txt*, and *telephone.txt*.

The output also shows that the *base16* scanner is a recursive scanner (indicated by the flag SCANNER_RECURSE) meaning it expands data or finds new data for other scanners to process. It also writes to the file *hex.txt*.

Finally, the output shows that the *wordlist* scanner is disabled by default (indicated by the flag SCANNER_DISABLED). This means that if the user would like to use the *wordlist* scanner, it will have to be specifically enabled. The wordlist scanner is useful for password cracking and is discussed in **Subsection 5.4**.

In general, most users will not need to enable or disable scanners. The default settings installed with the *bulk_extractor* system work best for the majority of users. However, individual scanners can be enabled or disabled for different purposes. To enable the *wordlist* scanner, which is disabled by default, use the following command:

```
■ bulk_extractor -e wordlist -o output diskimage.raw
```

Additionally, users can disable a scanner that is enabled by default. Most of the scanners are enabled by default. To disable the *accts* scanner, which is very CPU intensive, run the following command:

```
■ bulk_extractor -x accts -o output diskimage.raw
```

The command -E disables all scanners, then enables the one that follows the option. For example, to disable all scanners except the *aes* scanner, use the following command:

```
■ bulk_extractor -E aes -o output diskimage.raw
```

The options -E, -e and -x are all processed in order. So, the following command will also disable all scanners and then enable the *aes* scanner:

```
■ bulk_extractor -x all -e aes -o output diskimage.raw
```

Some of the scanners installed with *bulk_extractor* have parameters that can be set and utilized by advanced users for different purposes. Those parameters are also described in the -H output described above (as well as the -h output) and include the following:

Settable Options (and their defaults):

```
-S work_start_work_end=YES      Record work start and end of each scanner in report.xml file ()
-S enable_histograms=YES       Disable generation of histograms ()
-S debug_histogram_malloc_fail_frequency=0  Set >0 to make histogram maker fail with memory allocations
-S hash_alg=md5                Specifies hash algorithm to be used for all hash calculations ()
-S dup_data_alerts=NO          Notify when duplicate data is not processed ()
-S write_feature_files=YES      Write features to flat files ()
-S write_feature_sqlite3=NO     Write feature files to report.sqlite3 ()
-S report_read_errors=YES      Report read errors ()
-S ssn_mode=0                  0=Normal; 1=No 'SSN' required; 2=No dashes required (accts)
-S min_phone_digits=6          Min. digits required in a phone (accts)
-S carve_net_memory=NO         Carve network memory structures (net)
-S word_min=6                  Minimum word size (wordlist)
-S word_max=14                 Maximum word size (wordlist)
-S max_word_outfile_size=100000000 Maximum size of the words output file (wordlist)
-S wordlist_use_flatfiles=NO    Override SQL settings and use flatfiles for wordlist (wordlist)
-S hashdb_mode=none            Operational mode [none|import|scan]
    none      - The scanner is active but performs no action.
    import    - Import block hashes.
    scan      - Scan for matching block hashes. (hashdb)
-S hashdb_block_size=4096      Hash block size, in bytes, used to generate hashes (hashdb)
-S hashdb_ignore_empty_blocks=YES Selects to ignore empty blocks. (hashdb)
-S hashdb_scan_path_or_socket=your_hashdb_directory File path to a hash database or
    socket to a hashdb server to scan against. Valid only in scan mode. (hashdb)
-S hashdb_scan_sector_size=512 Selects the scan sector size. Scans along
    sector boundaries. Valid only in scan mode. (hashdb)
-S hashdb_import_sector_size=4096 Selects the import sector size. Imports along
    sector boundaries. Valid only in import mode. (hashdb)
-S hashdb_import_repository_name=default_repository Sets the repository name to
    attribute the import to. Valid only in import mode. (hashdb)
-S hashdb_import_max_duplicates=0 The maximum number of duplicates to import
    for a given hash value, or 0 for no limit. Valid only in import mode. (hashdb)
-S exif_debug=0               debug exif decoder (exif)
-S jpeg_carve_mode=1           0=carve none; 1=carve encoded; 2=carve all (exif)
-S min_jpeg_size=1000          Smallest JPEG stream that will be carved (exif)
-S zip_min_uncompr_size=6      Minimum size of a ZIP uncompressed object (zip)
-S zip_max_uncompr_size=268435456 Maximum size of a ZIP uncompressed object (zip)
-S zip_name_len_max=1024       Maximum name of a ZIP component filename (zip)
-S unzip_carve_mode=1           0=carve none; 1=carve encoded; 2=carve all (zip)
-S rar_find_components=YES     Search for RAR components (rar)
```

```

-S raw_find_volumes=YES      Search for RAR volumes (rar)
-S unrar_carve_mode=1        0=carve none; 1=carve encoded; 2=carve all (rar)
-S gzip_max_uncompr_size=268435456    maximum size for decompressing GZIP objects (gzip)
-S pdf_dump=NO              Dump the contents of PDF buffers (pdf)
-S opt_weird_file_size=157286400    Weird file size (windirs)
-S opt_weird_file_size2=536870912    Weird file size2 (windirs)
-S opt_max_cluster=67108864    Ignore clusters larger than this (windirs)
-S opt_max_cluster2=268435456    Ignore clusters larger than this (windirs)
-S opt_max_bits_in_attr=3      Ignore FAT32 entries with more attributes set than this (windirs)
-S opt_max_weird_count=2      Ignore FAT32 entries with more things weird than this (windirs)
-S opt_last_year=2019         Ignore FAT32 entries with a later year than this (windirs)
-S xor_mask=255              XOR mask string, in decimal (xor)
-S sqlite_carve_mode=2        0=carve none; 1=carve encoded; 2=carve all (sqlite)

```

To use any of these options, the user should specify the `-S` with the name=value pair when running *bulk_extractor* as in the following example:

```
■ bulk_extractor -S name=value -o output diskimage.raw
```

As with the other scanner and *bulk_extractor* usage options, most users will not have to use any of these options.

4.3 Carving

File carving is a special kind of carving in which files are recovered. 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 [?]. Currently, *bulk_extractor* provides carving of contiguous JPEG, ZIP and RAR files. To carve fragmented files we recommend **PhotoRec** (free) or **Adroit Photo Recovery** (commercial). Additionally, **Forensics Toolkit** and **EnCase Forensic** provide some carving capability on fragmented files.

Carved results are stored in two different places. First, a file listing all the files that are carved are written to a corresponding .txt file: JPEG files to `jpeg.txt`, ZIP files to `unzip.txt` and RAR files to `unrar.txt`. Second, the carved JPEG, ZIP and RAR files are placed in binned directories that are named `/jpeg`, `/unzip` and `/unrar` respectively. For example, all carved JPEGs will go in the directory `/jpeg`. The output files are further binned with 1000 files in each directory. The directory names are 3 decimal digits. If there are more than 999,000 carved files of one type, then the next set of directories are named with 4 digits. File names for JPEGs are the `forensicpath.jpg`. File names for the ZIP carver are the `forensicpath_filename`. If the ZIP file name has slashes in it (denoting directories), they are turned into `'_'` (underbars). For example, the file `mydocs/output/specialfile` will be named `mydocs_output_specialfile`.

As the above table describes, there are three carving modes in *bulk_extractor* that can be specified separately for each file type, JPEG, ZIP or RAR. The first mode, mode 0, explicitly tells *bulk_extractor* not to carve files of that type. The second mode, mode 1, is on by default and tells *bulk_extractor* to carve only encoded files of that type. If the user is running the ZIP carver in mode 1 and there is a simple ZIP file, it will not get carved. However, if there is an encoded attachment of that file (like Base64) it will get carved. The final mode, mode 2, will carve everything of that type. There is no way to specify which types of files (particular extensions) will get carved and which will not in mode 2. For example, *bulk_extractor* will carve both JPEGs and doc files. It carves

whatever is encountered.

To specify the carving modes for *bulk_extractor*, command line arguments can be specified. To modify the JPEG carving modes, type the following where carve mode 1=default value that does not need to be specified (carve encoded), 0=no carving or 2=carve everything:

```
■ bulk_extractor -S jpeg_carve_mode=1 -o output diskimage.raw
```

To modify the ZIP carving modes, type the following where carve mode 1=default value that does not need to be specified (carve encoded), 0=no carving or 2=carve everything:

```
■ bulk_extractor -S zip_carve_mode=1 -o output diskimage.raw
```

To modify the RAR carving modes, type the following where carve mode 1=default value that does not need to be specified (carve encoded), 0=no carving or 2=carve everything:

```
■ bulk_extractor -S rar_carve_mode=1 -o output diskimage.raw
```

Any combination of the carving mode options can be specified for a given run. The carvers can run in any combination of modes. For example, the JPEG carver can be run in mode 2 while the RAR carving is turned off in mode 1 and the ZIP carver carves only encoded files in mode 1.

Because *bulk_extractor* can carve files and preserve original file extensions, there is a real possibility that *bulk_extractor* might be carving out malware. There is no protection in *bulk_extractor* against putting malware in a file on your hard drive. Users running *bulk_extractor* to look for malware should turn off all anti-virus software because the anti-virus program will think its creating malware and stop it. Then the user should carefully scan the results looking for malware before re-enabling the anti-virus.

4.4 Suppressing False Positives

Modern operating systems are filled with email addresses. They come from Windows binaries, SSL certificates and sample documents. Most of these email addresses, particularly those that occur the most frequently, such as `someone@example.com`, are not relevant to the case. It is important to be able to suppress those email addresses not relevant to the case. To address this problem, *bulk_extractor* provides two approaches.

First, *bulk_extractor* allows users to build a stop list or use an existing one available for download. These stop lists are used to recognize and dismiss the email addresses that are native to the Operating System. This approach works well for email addresses that are clearly invalid, such as `someone@example.com`. For most email addresses, however, you will want to stop them in some circumstances but not others. For example, there are over 20,000 Linux developers, you want to stop their email addresses in program binaries, not in email messages.

To address this problem, *bulk_extractor* uses context-sensitive stop lists. Instead of a stop list of features, this approach uses the feature+context. The following example is an excerpt from a context-sensitive stop list file.

Without Stop List		With Stop List
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@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 49091023.607@gmail.com

Figure 14: Email Histogram Results With and Without the Context-Sensitive Stop List. Results from the Domexusers HD image.

```

ubuntu-users@lists.ubuntu.com      Maint\x0A935261357\x09ubuntu-users@lists.ubuntu.com\x0
ubuntu-motu@lists.ubuntu.com      untu_\x0A923867047\x09ubuntu-motu@lists.ubuntu.com\x09
pschiffe@redhat.com      Peter Schiffer <pschiffe@redhat.com> - 0.8-1.1N\x94/\xC0-
phpdevel@echospace.com      : Vlad Krupin <phpdevel@echospace.com>\x0AMAINTEANCE:
anholt@freebsd.org      34-GZIP-1021192\x09anholt@freebsd.org\x09r: EricAnholt
ubuntu-motu@lists.ubuntu.com      http\x0A938966489\x09ubuntu-motu@lists.ubuntu.com\x09

```

The context for the feature is the 8 characters on either side of the feature. Each “stop list” entry is the feature+context. This ignores Linux developer email addresses in Linux binaries. The email address will be ignored if found in that context but reported if it appears in a different context.

There is a context-sensitive stop list for Microsoft Windows XP, 2000, 2003, Vista and several Linux systems. The total stop list is 70 MB and includes 628,792 features in a 9 MB zip file. The context-sensitive stop list prunes many of the OS-supplied features. By applying it to the domexusers HD image (the image can be downloaded at <http://http://digitalcorpora.org/corp/nps/drives/nps-2009-domexusers/>, the number of emails found went from 9,143 down to 4,459. This significantly reduces the amount of work to be done by the investigator. Figure 14 shows how the histogram of email addresses differs when *bulk_extractor* is run with and without the context-sensitive stop list. The context-sensitive stop list built for the various operating systems described above can be downloaded from http://digitalcorpora.org/downloads/bulk_extractor. The file will have the words “stoplist” in it somewhere. The current version as of publication of this manual is called *bulk_extractor-3-stoplist.zip*.

It should be noted that *bulk_extractor* does allow the users to create stop lists that are not context sensitive. A stop list can simply be a list of words that the user wants *bulk_extractor* to ignore. For example, the following three lines would constitute a valid stop list file:


```
abc@google.com
ignore@microsoft.com
www.google.com
```

However for the reasons stated above, it is recommended that users rely on context-sensitive stop lists when available to reduce the time required to analyze the results of a *bulk_extractor* run.

Stopped results are not completely hidden from users. If stopped feature are discovered, they will be written to the appropriate category feature file with the extension `_stopped.txt`. For example, stopped domain names that are found in the disk image will be written to `domain_stopped.txt` in the output directory. The stopped files serve the purpose of allowing users to verify that *bulk_extractor* is functioning properly and that the lists they have written are being processed correctly.

4.5 Using an Alert List

Users may have specific words, or feature in a given context, that are important to their investigation. The alert list allows *bulk_extractor* to specifically alert or flag the user when those concepts are found. Alert lists can contain a list of words or a feature file. The feature file operates much in the same way as the feature files used for context-sensitive stop lists. It will provide a feature but alert on that feature only when it's found in the specified context.

A sample alert list file might look like the following:

```
abc@google.com
SilentFury2012
www.maliciousintent.com
```

While this list does not appear to help in any particular investigation, it demonstrates that you can specify distinct words that are important to their analysis. Results containing the alert list information are found in the file `alert.txt` in the run output directory.

4.6 The Importance of Compressed Data Processing

Many forensic tools frequently miss case-critical data because they do not examine certain classes of compressed data found. For example, a recent study of 1400 drives found thousands of email addresses that were compressed but in unallocated space[?]. Without looking at all the data on each drive and optimistically decompressing it, those features would be missed. Compressed email addresses, such as those in a GZIP file, do not look like email addresses to a scanner; they must first be decompressed to be identified. Although some of these features are from software distributions, many are not. Table 3 shows the kinds of encodings that can be decoded by *bulk_extractor* [?].

The reason that users must be aware of this is because users have a tendency to want to enable and disable scanners for specific uses; They can unintentionally damage their results. For example, if a user only wants to find email addresses, they may try to turn off all scanners except the email scanner. This will find some email addresses. However, it will miss the email addresses on the media that are only present in compressed data. This is because scanners such as *zip*, *rar* and *gzip* will not be running. Those scanners

each work on a different type of compressed data. For example, the *gzip* scanner will find GZIP compressed data, decompress it and then pass it other scanners to search for features. In that way, GZIP compressed emails can be processed by *bulk_extractor*.

The *pdf* scanner is another type of scanner that finds text that otherwise wouldn't be found. While PDF files are human readable, they are not readable but many software tools and scanners because of their formatting. The *pdf* scanner extracts some kinds of text found within PDFs and then passes that text on to other scanners for further processing. Many typical disk images include PDF files, so most users will want to have this scanner enabled (as it is by default).

Finally, the *hiber* scanner decompresses Windows hibernation files. If the disk image being analyzed is from a Windows system, *bulk_extractor* users will want that turned on (as it is by default). The scanner is very fast, however, so it will not significantly decrease performance on non-Windows drives.

5 Use Cases for *bulk_extractor*

There are many digital forensic use cases for *bulk_extractor*— more than we can enumerate within this manual. In this section we highlight some of the most common uses of the system. Each case discusses which output files, including feature files and histograms, are most relevant to these types of investigations. In **Section 8, Worked Examples**, we provide more detailed walk-throughs and refer back to these use cases with more detailed output file information.

5.1 Malware Investigations

Malware is a programmatic intrusion. When performing a malware investigation, users will want to look at executables, information that has been downloaded from web-based applications and windows directory entries (for Windows-specific investigations). *bulk_extractor* enables this in several ways.

First, *bulk_extractor* finds evidence of virtually all executables on the hard drive including those by themselves, those contained in ZIP files, and those that are compressed. It does not give you the hash value of the full file, rather, it gives the hash of just the first 4KB of the file. Our research has shown that the first 4KB is predictive because most executables have a distinct hash value for the first 4KB of the file [?]. Additionally, many of these files may be fragmented and looking at the first 4KB will still provide information relevant to an investigation because fragmentation is unlikely to happen before the first 4KB. The full hash of a fragmented file is not available in *bulk_extractor*.

Several output feature files produced by *bulk_extractor* contain relevant and important information about executables. These files include:

- **elf.txt** — This file (produced by the *elf* scanner) contains information about ELF executables that can be used to target Linux and Mac systems.
- **winprefetch.txt** — This file (produced by the *winprefetch* scanner) lists the current and deleted files found in the Windows prefetch directory.

The XML in these feature files is too complicated to review without using other applications. The recommended way to analyze the executable output is to use a third party tool that analyzes executables or pull the results into a spreadsheet. In a spreadsheet, one column could contain the hash values and those values can be compared against a database of executable hashes. There is also a python tool that comes with *bulk_extractor* called **identify_filenames.py** that can be used to get the full filename of the file. The python tool is discussed in more detail in **Section 7**.

For Windows specific malware investigations, the files **winpe.txt** and **winprefetch.txt** are very useful. They are produced by the *winpe* and *winprefetch* scanners respectively. Windows Prefetch shows files that have been prefetched in the Windows prefetch directory and shows the deleted files that were found in unallocated space. The Windows PE feature file shows entries related to the Windows executable files.

JSON, the JavaScript Object Notation, is a lightweight data-interchange format. Websites tend to download a lot of information using JSON. The output file **json.txt**, produced by the *json* scanner, can be useful for malware investigations and analysis of web-based applications. If a website has downloaded information in JSON format, the JSON scanner may find that information in the browser cache.

5.2 Cyber Investigations

Cyber investigations may scan a wide variety of information types. A few unifying features of these investigations are the need to find encryption keys, hash values and information about ethernet packets. *bulk_extractor* provides several scanners that produce feature files containing this information.

For encryption information, the following feature files may be useful:

- **aes.txt** — AES is an encryption system. Many implementations leave keys in memory that can be found using an algorithm invented at Princeton University. *bulk_extractor* provides an improved version of that algorithm to find AES keys in the *aes* scanner. When it scans memory, such as swap files or decompressed hibernation files, it will identify the AES keys. The keys can be used for software that will decrypt AES encrypted material.
- **hex.txt** — The *base16* scanner decodes information that is stored in Base16, breaking it into the corresponding hexadecimal values. This is useful if you are looking for AES keys or SHA1 hashes. This scanner only writes blocks that are of size 128 and 256 because they are the sizes used for encryption keys. The feature file is helpful if the investigator is looking for people who have emailed encryption keys or hash values in a cyber investigation.

Additionally, the *base64* scanner is important for cyber investigations because it looks mostly at email attachments that are coded in Base64. The information found in these attachments will be analyzed by other scanners looking for specific features.

The *windirs* scanner finds Windows FAT32 and NTFS directory entries and will also be useful for cyber investigations involving Windows machines, as they may be indicators of times that activity took place.

Finally, the files `ether.txt`, `ip.txt`, `tcp.txt` and `domain.txt` are all produced by the *net* scanner. It searches for ethernet packets and memory structures associated with network data structures in memory. It is important to note that tcp connections have a lot of false positives and many of the information found by this scanner will be false. Investigators should be careful with the interpretation of these feature files for that reason.

5.3 Identity Investigations

Identity investigations may be looking for a wide variety of information including email addresses, credit card information, telephone numbers, geographical information and keywords. For example, if the investigator is trying to find out of who a person is and who their associates are, they will be looking at phone numbers, search terms to see what they are doing and emails to see who they are communicating with.

The *accts* scanner is very useful for identity investigations. It produces several feature files with identity information including:

- `ccn.txt` — credit card numbers
- `ccn_track2.txt` - credit card track two information - relevant information if someone is trying to make physical fake credit cards
- `pii.txt` - personally identifiable information including birth dates and social numbers
- `telephone.txt` - telephone numbers

The *kml* and *gps* scanner both produce GPS information that give information about a person in a certain area or link to what they have been doing in a certain area. Both of these scanners write to `gps.txt`. KML is a format used by Google Earth and Google Map files. This scanner searches in that formatted data for GPS coordinates. The *gps* scanner looks at Garmin Trackpoint formatted information and finds GPS coordinates in that data.

The *email* scanner looks for email addresses in all data and writes that to `email.txt`. The *vcards* scanner looks at vCard data, an electronic business card format, and finds names, email addresses and phone numbers to write to the respective feature file.

There are multiple url files including `url.txt`, `url_facebook-address`, `url_facebook-id`, `url_microsoft-live`, `url_searches.txt` and `url_services.txt` that are produced by the *email* scanner. They are useful for looking at what websites a person has visited as well as the people they are associating with.

An important aspect of identity investigations (as well as other types) is the ability to search the data for a list of keywords. *bulk_extractor* provides the capability to do that through two different means. First, the *find* scanner is a simple regular expression finder that uses regular expressions. The *find* scanner looks through the data for anything listed in the global find list. The format of the find list should be rows of regular expressions while any line beginning with a `#` is considered a comment. The following is an excerpt from a sample find list file:

```
# This is a comment line
\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b
# another comment line
/^[a-z0-9_-]{3,16}$/
```

The first regular expression from the above example, beginning with `\b`, looks for the following in order: a word boundary followed a digit repeated between 1-3 times, a digit repeated between 1-3 times, a digit repeated 1-3 times, a '.', a digit repeated 1-3 times, a digit repeated 1-3 times and the end of the word boundary. That regular expression would find, for example, the sequence 2219.889 separated out from other text by a word boundary.

The second regular expression from the above example, beginning with `/` looks for the following in order: a '/', the beginning of a line, repeats of any character in lowercase a-z, 0-9, '_', or '-', repeated 3 to 16 times, and the end of the line followed by '\.' That expression would find, for example, the following sequence:

```
\
284284284284
/
```

Regular expressions can be used to represent character and number sequences (or ranges of values) that might be of particular importance to an investigation.

The find list is sent in as input to *bulk_extractor* using the “-F findlist” option. To run *bulk_extractor* with a find list, the following basic parameters are required (where *findlist.txt* is the name of the find list):

```
■ bulk_extractor -F findlist.txt -o output mydisk.raw
```

Another scanner, the *lightgrep* scanner provides the same functionality as the *find* scanner but it is much faster and provides more functionality. It is also a regular expression scanner that looks through the buffers and matches in the global find list. A syntax sheet of regular expressions that might be helpful to users in creating a find list to be used by the Lightgrep Scanner is shown in Figure 15.

The *lightgrep* scanner uses the Lightgrep library from Lightbox Technologies. An open source version of that library can be downloaded from <https://github.com/LightboxTech/liblightgrep>. Installation instructions are also available at the download site. The *lightgrep* scanner is preferable because it looks for all regular expressions at once, on the first pass through the data. The *find* scanner actually looks for each expression in the find list one at a time. For example, if the find list is a list of medical terms and diagnoses and *bulk_extractor* is searching medical records, the *find* scanner looks for each term in each piece of data on one pass through, one at a time. A list of 35 expressions would require 35 passes through the data. The *lightgrep* scanner will search a given buffer for all of the medical terms at once, in one pass through.

If the Lightgrep library is installed and the find list is provided to *bulk_extractor*, it will run the *lightgrep* scanner. If not, it will use the *find* scanner. Neither scanner needs to be enabled by the user specifically, calling *bulk_extractor* with the find list will automatically enable the appropriate scanner. However, we do not recommend using the find list without the Lightgrep library — it will make *bulk_extractor* run very slowly because each find search will be sequentially executed. This will provide an exponential

1 Single Characters

<code>c</code>	the character <code>c*</code>
<code>\a</code>	U+0007 (BEL) bell
<code>\e</code>	U+001B (ESC) escape
<code>\f</code>	U+000C (FF) form feed
<code>\n</code>	U+000A (NL) newline
<code>\r</code>	U+000D (CR) carriage return
<code>\t</code>	U+0009 (TAB) horizontal tab
<code>\ooo</code>	U+ooo, 1-3 octal digits <i>o</i> , ≤ 0377
<code>\xhh</code>	U+00hh, 2 hexadecimal digits <i>h</i>
<code>\x{hhhhhh}</code>	U+hhhhhh, 1-6 hex digits <i>h</i>
<code>\zh</code>	the byte 0xhh (not the character!)*
<code>\N{name}</code>	the character called <i>name</i>
<code>\N{U+hhhhhh}</code>	same as <code>\x{hhhhhh}</code>
<code>\c</code>	the character <code>c†</code>

*except U+0000 (NUL) and metacharacters
†Lightgrep extension; not part of PCRE.
‡except any of: adefnrstwdPSW1234567890

2 Named Character Classes

<code>.</code>	any character
<code>\d</code>	[0-9] (= ASCII digits)
<code>\D</code>	[^0-9]
<code>\s</code>	[\t\n\f\r] (= ASCII whitespace)
<code>\S</code>	[^\t\n\f\r]
<code>\w</code>	[0-9A-Za-z_] (= ASCII words)
<code>\W</code>	[^0-9A-Za-z_]
<code>\p{property}</code>	any character having <i>property</i>
<code>\P{property}</code>	any character lacking <i>property</i>

3 Character Classes

<code>[stuff]</code>	any character in <i>stuff</i>
<code>[^stuff]</code>	any character not in <i>stuff</i>

where *stuff* is...

<code>c</code>	a character
<code>a-b</code>	a character range, inclusive
<code>\zh</code>	a byte
<code>\zhh-\zh</code>	a byte range, inclusive
<code>[S]</code>	a character class
<code>ST</code>	$S \cup T$ (union)
<code>S&&T</code>	$S \cap T$ (intersection)
<code>S-T</code>	$S - T$ (difference)
<code>S~T</code>	$S \Delta T$ (symmetric difference, XOR)

4 Grouping

(*S*) makes any pattern *S* atomic

5 Concatenation & Alternation

<code>ST</code>	matches <i>S</i> , then matches <i>T</i>
<code>S T</code>	matches <i>S</i> or <i>T</i> , preferring <i>S</i>

6 Repetition

<code>S*</code>	Repeats <i>S</i> ...
<code>S+</code>	0 or more times (= $S\{0, \}$)
<code>S?</code>	1 or more times (= $S\{1, \}$)
<code>S{n}</code>	0 or 1 time (= $S\{0, 1\}$)
<code>S{n,m}</code>	<i>n</i> or more times
<code>S{n,m}</code>	<i>n-m</i> times, inclusive

7 Selected Unicode Properties

Any	Assigned
Alphabetic	White_Space
Uppercase	Lowercase
ASCII	Noncharacter_Code_Point
Name=name	Default_Ignorable_Code_Point

General_Category=category

L, Letter	P, Punctuation
Lu, Uppercase Letter	Pc, Connector Punctuation
Ll, Lowercase Letter	Pd, Dash Punctuation
Lt, Titlecase Letter	Ps, Open Punctuation
Lm, Modifier Letter	Pe, Close Punctuation
Lo, Other Letter	Pi, Initial Punctuation
M, Mark	Pf, Final Punctuation
Mn, Non-Spacing Mark	Po, Other Punctuation
Me, Enclosing Mark	Z, Separator
N, Number	Zs, Space Separator
Nd, Decimal Digit Number	Zl, Line Separator
Nl, Letter Number	Zp, Paragraph Separator
No, Other Number	C, Other
S, Symbol	Cc, Control
Sm, Math Symbol	Cf, Format
Sc, Currency Symbol	Cs, Surrogate
Sk, Modifier Symbol	Co, Private Use
So, Other Symbol	Cn, Not Assigned

Script=script

Common Latin Greek Cyrillic Armenian Hebrew Arabic Syraic Thaana Devanagari Bengali Gurmukhi Gujarati Oriya Tamil Telugu Kannada Malayalam Sinhala Thai Lao Tibetan Myanmar Georgian Hangul Ethiopic Cherokee Ogham Runic Khmer Mongolian Hiragana Katakana Bopomofo Han Yi Old Italic Gothic Inherited Tagalog Hanunoo Buhid Tagbanwa Limbu Tai_Le Linear_B Ugaritic Shavian Osmanya Cypriot Buginese Coptic New_Tai_Lue Glagolitic Tifinagh Syloti_Nagri Old_Persian Kharoshthi Balinese Cuneiform Phoenician Phags_Pa Nko Sudanese Lepcha ... See Unicode Standard for more.

8 EnCase GREP Syntax

<code>c</code>	the character <i>c</i> (except metacharacters)
<code>\xhh</code>	U+00hh, 2 hexadecimal digits <i>h</i>
<code>\whhhh</code>	U+hhhhh, 4 hexadecimal digits <i>h</i>
<code>\c</code>	the character <i>c</i>
<code>.</code>	any character
<code>[0-9]</code>	[0-9] (= ASCII digits)
<code>[a-b]</code>	any character in the range <i>a-b</i>
<code>[S]</code>	any character in <i>S</i>
<code>[^S]</code>	any character not in <i>S</i>
<code>(S)</code>	grouping
<code>S*</code>	repeat <i>S</i> 0 or more times (max 255)
<code>S+</code>	repeat <i>S</i> 1 or more times (max 255)
<code>S?</code>	repeat <i>S</i> 0 or 1 time
<code>S{n,m}</code>	repeat <i>S</i> <i>n-m</i> times (max 255)
<code>ST</code>	matches <i>S</i> , then matches <i>T</i>
<code>S T</code>	matches <i>S</i> or <i>T</i>

9 Importing from EnCase into Lightgrep

<code>\whhhh</code>	→ <code>\xhhhh</code>	<i>S*</i> and <i>S+</i> are limited to 255 repetitions by EnCase;
<code>#</code>	→ <code>\d</code>	Lightgrep preserves this in
<i>S*</i>	→ <i>S{0, 255}</i>	imported patterns.
<i>S+</i>	→ <i>S{1, 255}</i>	

\w is limited to BMP characters (≤ U+10000) only.

Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.
—JWZ in alt.religion.emacs, 12 August 1997

Lightgrep Cheat Sheet

Notes & Examples

Characters:

- `.*?\x00` (= null-terminated string)
- `\z50\z4B\z03\z04` (= ZIP signature)
- `\N{EURO SIGN}, \N{NO-BREAK SPACE}`
- `\x{042F}` (= CYRILLIC CAPITAL LETTER YA)
- `\+12\.%` (= escaping metacharacters)

Grouping: Operators bind tightly. Use `(aa)+`, not `aa+`, to match pairs of a's.

Ordered alternation: `a|ab` matches a twice in `aab`. Left alternatives preferred to right.

Repetition: Greedy operators match as much as possible. Reluctant operators match as little as possible. `a+a` matches all of `aaaa`; `a+?a` matches the first `aa`, then the second `aa`.

`+` will (uselessly) match the **entire** input. Prefer reluctant operators when possible.

Character classes:

- `[abc]` = a, b, or c
- `[^a]` = anything but a
- `[A-Z]` = A to Z
- `[A\ -Z]` = A, Z, or hyphen (!)
- `[A-Zaeiou]` = capitals or lowercase vowels
- `[. +* \]`
- `[^ . +* \]`
- `[Q\z00-\z7F]` = Q or 7-bit bytes
- `[[abcd][bce]]` = a, b, c, d, or e
- `[[abcd]&&[bce]]` = b or c
- `[[abcd]-[bce]]` = a or d
- `[[abcd]~[bce]]` = a, d, or e
- `[\p{Greek}\d]` = Greek or digits
- `[\^p{Greek}7]` = neither Greek nor 7
- `[\p{Greek}&&\p{Ll}]` = lowercase Greek

Operators need not be escaped inside character classes.

Lightgrep Search for EnCase®

Fast Search for Forensics

www.lightgrep.com

Email addresses: `[a-z\d!#$%&'*/=?^_`{|}~][a-z\d!#$%&'*/=?^_`{|}~]{0,63}@[a-z\d.-]{1,253}\.[a-z\d-]{2,22}`

Hostnames: `(([a-z\d]([a-z\d-]{0,61}[a-z\d])?)\.[a-z\d-]{2,22}){1,253}`

N. American phone numbers: `\(?\d{3}[]?\d{0,2}\d{3}[]?\d{0,2}\d{3}[]?\d{0,2}\d{3}\)`

Visa, MasterCard: `\d{4}([]?\d{4}){3}`

American Express: `3[47]\d{2}([]?\d{6}[]?\d{5})`

Diners Club: `3[08]\d{2}([]?\d{6}[]?\d{5})`

EMF header: `\z01\z00\z00\z00.{36}\z20EMF`

JPEG: `\zFF\zD8\zFF[\zC4\zDB\zE0-\zEF\zFE]` Footer: `\zFF\zD9`

GIF: `GIF8[79]` Footer: `\z00\z3B` BMP: `BM.{4}\z00\z00\z00\z00.{4}\z28`

PNG: `\z89\z50\z4E\z47` Footer: `\z49\z45\z4E\z44\zAE\z42\z60\z82`

ZIP: `\z50\z4B\z03\z04` Footer: `\z50\z4B\z05\z06`

RAR: `\z52\z61\z72\z21\z1a\z07\z00...[\z00-\z7F]`

Footer: `\z88\zC4\z3D\z7B\z00\z40\z07\z00`

GZIP: `\z1F\z8B\z08` MS Office 97-03: `\zD0\zCF\z11\zE0\zA1\zB1\z1A\zE1`

LNK: `\z4c\z00\z00\z00\z01\z14\z02\z00`

PDF: `\z25\z50\z44\z46\z2D\z31` Footer: `\z25\z45\z4F\z46`

Figure 15: Guide to Syntax Used by Lightgrep Scanner

31

slow-down.

Investigators looking for identity information may rely heavily on the find list to search for specific names, numbers or keywords relevant to the investigation. The features found by the *find* or *lightgrep* scanner will be written to the files `find.txt` and `lightgrep.txt` respectively.

5.4 Password Cracking

If an investigator is looking to crack a password, the *wordlist* scanner can be useful. It generates a list of all the words found on the disk that are between 6 and 14 characters. Users can change the minimum and maximum size of words by specifying options at run-time but we have found this size range to be optimal for most applications. Because the *wordlist* scanner is disabled by default, users must specifically enable it at run-time when needed. To do that, run the following command:

```
■ bulk_extractor -e wordlist -o output mydisk.raw
```

This will produce two files useful for password cracking, `wordlist_histogram.txt` and `wordlist.txt`. These files will contain large words that can be used to recommend passwords.

5.5 Analyzing Imagery Information

In an investigator needs to specifically analyze imagery, for something such as a child pornography investigation, the *exif* scanner would be useful. It finds JPEGs on the disk image and then carves the encoded ones that might be in, for example, ZIP files or hibernation files. It writes the output of this carving to `jpeg.txt`.

5.6 Using *bulk_extractor* in a Highly Specialized Environment

If using *bulk_extractor* in a specialized environment, two specific features might be useful. The first is the option to include a banner on each output file created by *bulk_extractor*. The banner file, specified in the example command below as `banner.txt` could include a security classification of the output data. When *bulk_extractor* is run with the command specified below, the data in the banner file will be printed at the top of each output file produced.

```
■ bulk_extractor -b banner.txt -o output mydisk.raw
```

The second feature might be useful to users in a specialized environment is the ability to develop plug-ins. Plug-ins in *bulk_extractor* are external scanners that an individual or organization can run in addition to the open source capabilities provided with the *bulk_extractor* system. The plug-in system gives the full power of *bulk_extractor* to external developers, as all of *bulk_extractor*'s native scanners are written with the plug-in system. This power gives third party developers the ability to utilize proprietary or security protected algorithms and information in *bulk_extractor* scanners. It is worth noting that all scanners installed with *bulk_extractor* use the plug-in system, *bulk_extractor* is really just a framework for running plug-ins. The separate publication **Programmers Manual for Developing Scanner Plug-ins** [?] provides specific details on how to develop and use plug-ins with *bulk_extractor*.

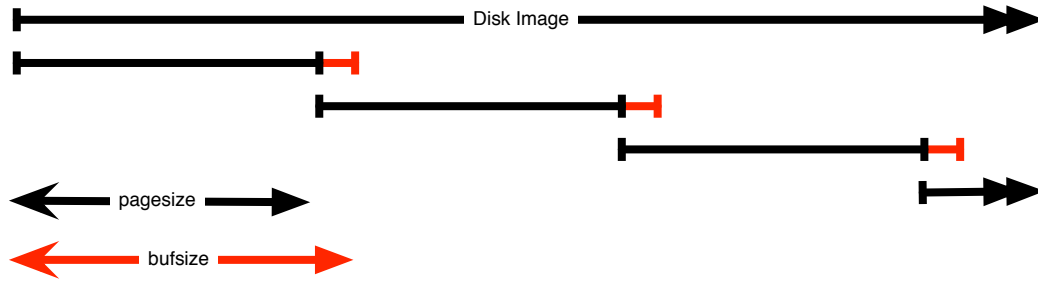


Figure 16: Image Processor divides the disk image into buffers. Each buffer is the size of a page (*pagesize*) with a buffer overlap in an area called the margin. (*margin*size is equal to *bufsize*-*pagesize*). The buffers overlap with each other to ensure all information is processed.

6 Tuning *bulk_extractor*

All data that *bulk_extractor* processes is divided into buffers called sbufs. Buffers created from disk images are created with a pre-determined size (*bufsize*). The buffer includes a page and an overlap area. As shown in Figure 16, the pages overlap with each other in the red region. The red overlap region is called the margin. *bulk_extractor* scans the pages one-by-one looking for features. Pages overlap with each other so that *bulk_extractor* won't miss any features that cross from one page into another across boundaries.

Users may be looking for potentially large features that are bigger than the buffer size or that overlap into the margin. In that case, they may want to adjust the margin size or buffer size. For example, if the input data includes a 30 MB ZIP file (possibly a software program), *bulk_extractor* won't find features in the program because it overlaps the margins. To find features of that size, the margin size must be increased.

To adjust the page size, the following usage options need to be included where NN should be set to the size (default page size is 16777216):

```
■ bulk_extractor -G NN -o output mydisk.raw
```

To adjust the margin size, the following usage options need to be included where NN should be set to the size (default margin size is 4194304):

```
■ bulk_extractor -g NN -o output mydisk.raw
```

bulk_extractor provides many other tuning capabilities that are primarily recommended for users doing advanced research. Many of those options relate to specifying file sizes for input or output, specifying block sizes, dumping the contents of a buffer or ignoring certain entries. Those options are all found in the output of the -h input to *bulk_extractor* and listed in **Appendix A**.

7 Post Processing Capabilities

There are two Python programs useful for post-processing the *bulk_extractor* output. Those programs are **bulk_diff.py** and **identify_filenames.py**. To run either of these programs, you must have Python version 2.7 or higher installed on your system. On

Linux and Mac systems, the *bulk_extractor* python programs are located in the directory *./python* under the main *bulk_extractor* installation.

7.1 **bulk_diff.py**: Difference Between Runs

The program **bulk_diff.py** takes the results of two *bulk_extractor* runs and shows the differences between the two runs. This program essentially tells the difference between two disk images. It will note the different features that are found by *bulk_extractor* between one image and the next. It can be used, for example, to easily tell whether or not a computer user has been visiting websites they are not supposed to by comparing a disk image from their computer from one week to the next. To run the program, users should type the following, where *pre* and *post* are both locations of two *bulk_extractor* output directories:

■ **bulk_diff.py** <pre> <post>

Note, Linux and Mac users may have to type **python2.7**, **python3**, or **python3.3** before the command, indicating the version of Python installed on your machine. An example use of the **bulk_diff.py** program can be found in Section 8.

7.2 **identify_filenames.py**: Identify File Origin of Features

The program **identify_filenames.py** operates on the results of *bulk_extractor* run and identifies the filenames (where possible) of the features that were found on the disk image. The user can run this program on one or all of the features file produced by a given run. It can be used, for example, to find the full content of an email when references to its contents are found in one of the feature files. Often email features are relevant to an investigation and an investigator would like to be able to view the full email.

To run this program, users will need the program **fiwalk** installed on their machine or have a DFXML file generated by **fiwalk** that corresponds to the disk image. **fiwalk** is part of the **SleuthKit** and can be installed by installing **Sleuthkit**, available at <http://www.sleuthkit.org/>.

The **identify_filenames.py** program provides various usage options but to run the program on all feature files produced by a *bulk_extractor* run, the user should type the following (where “bulkoutputdirectory” is the directory containing the output of a *bulk_extractor* run and “idoutput” will contain the annotated feature files after the program runs):

■ **identify_filenames.py** --all bulkoutputdirectory idoutput

Note, Linux and Mac users may have to type **python2.7**, **python3**, or **python3.3** before the command, indicating the version of Python installed on your machine. An example use of the **bulk_diff.py** program can be found in Section 8.

8 Worked Examples

The worked examples provided are intended to further illustrate how to use *bulk_extractor* to answer specific questions and conduct investigations. Each example uses a different, publicly available dataset and can be replicated by readers of this manual.

8.1 Encoding

We describe the encoding system here in order to prepare users to view the feature files produced by *bulk_extractor*. Unicode is the international standard used by all modern computer systems to define a mapping between information stored inside a computer and the letters, digits, and symbols that are displayed on the screens or printed on paper. UTF-8 is a variable width encoding that can represent every character in the Unicode character set. It was designed for backward compatibility with ASCII and to avoid the complications of endianness and byte order marks in UTF-16 and UTF-32. Feature files in *bulk_extractor* are all coded in UTF-8 format. This means that the odd looking symbols, such as accented characters (è), funny symbols (.:) and the occasional Chinese character (愛) that may show up in the files are legitimate. Glyphs from language, for example, Cyrillic (Ш) or Arabic (ﻉ) may show up in features files as all foreign languages can be coded in UTF-8 format. It is perfectly appropriate and typical to open up a feature file and see characters that the user may not recognize.

9 2009-M57 Patents Scenario

The 2009-M57-Patents scenario tracks the first four weeks of corporate history of the (fictional) M57 Patents company. The company started operation on Friday, November 13th, 2009, and ceased operation on Saturday, December 12, 2009. This specific scenario was built to be used as a teaching tool both as a disk forensics exercise and as a network forensics exercise. The scenario data is also useful for computer forensics research because the hard drive of each computer and each computers memory were imaged every day. In this example, we are not particularly interested in the exercises related to illegal activity, exfiltration and eavesdropping; they do however provide interesting components for us to examine in the example data[?].

9.1 Run *bulk_extractor* with the Data

For this example, we downloaded and utilized one of the disk images from the 2009-M57-Patents Scenario. Those images are available at <http://digitalcorpora.org/corp/nps/scenarios/2009-m57-patents/drives-redacted/>. The file used throughout this example is called **charlie-2009-12-11.E01**. Running *bulk_extractor* on the command line produces the following output (text input by the user is bold):

```
C:\bulk_extractor>bulk_extractor -o ../Output/charlie-2009-12-11 charlie-2009-12-11.E01
bulk_extractor version: 1.4.0
Input file: charlie-2009-12-11.E01
Output directory: ../Output/charlie-2009-12-11
Disk Size: 10239860736
Threads: 4
8:02:08 Offset 67MB (0.66%) Done in 1:21:23 at 09:23:31
8:02:34 Offset 150MB (1.47%) Done in 1:05:18 at 09:07:52
8:03:03 Offset 234MB (2.29%) Done in 1:01:39 at 09:04:42
8:03:49 Offset 318MB (3.11%) Done in 1:09:19 at 09:13:08
...
9:06:23 Offset 10049MB (98.14%) Done in 0:01:13 at 09:07:36
9:06:59 Offset 10133MB (98.96%) Done in 0:00:41 at 09:07:40
9:07:29 Offset 10217MB (99.78%) Done in 0:00:08 at 09:07:37
```

All data are read; waiting for threads to finish...

```

Time elapsed waiting for 4 threads to finish:
    (timeout in 60 min .)
Time elapsed waiting for 3 threads to finish:
    7 sec (timeout in 59 min 53 sec.)
Thread 0: Processing 10200547328
Thread 2: Processing 10217324544
Thread 3: Processing 10234101760

Time elapsed waiting for 2 threads to finish:
    13 sec (timeout in 59 min 47 sec.)
Thread 0: Processing 10200547328
Thread 2: Processing 10217324544

```

```

All Threads Finished!
Producer time spent waiting: 3645.8 sec.
Average consumer time spent waiting: 3.67321 sec.

```

```

*****
** bulk_extractor is probably CPU bound. **
**   Run on a computer with more cores   **
**   to get better performance.         **
*****
Phase 2. Shutting down scanners
Phase 3. Creating Histograms
    ccn histogram...    ccn_track2 histogram...    domain histogram...
    email histogram...  ether histogram...    find histogram...
    ip histogram...    lightgrep histogram...    tcp histogram...
    telephone histogram...    url histogram...    url microsoft-live...
    url services...    url facebook-address...    url facebook-id...
    url searches...Elapsed time: 3991.77 sec.
Overall performance: 2.56524 MBytes/sec
Total email features found: 15277

```

All of the results from the *bulk_extractor* run are stored in the output directory *charlie-2009-12-11*. The contents of that directory after the run include the feature files, histogram files and carved output. Figure 17 is a screenshot of the Windows output directory. Additionally, the following output shows a list of the files, directories and their sizes under Linux:

```

C:\bulk_extractor\charlie-2009-12-11>ls -s -F
    1 aes_keys.txt                0 kml.txt
    0 alerts.txt                 0 lightgrep.txt
    4 ccn.txt                     0 lightgrep_histogram.txt
    1 ccn_histogram.txt           196 packets.pcap
    0 ccn_track2.txt              1 rar.txt
    0 ccn_track2_histogram.txt    108 report.xml
23028 domain.txt                 3728 rfc822.txt
   192 domain_histogram.txt       20 tcp.txt
    0 elf.txt                     4 tcp_histogram.txt
1696 email.txt                   60 telephone.txt
   36 email_histogram.txt         8 telephone_histogram.txt
   24 ether.txt                   70108 url.txt
    1 ether_histogram.txt         1 url_facebook-address.txt
   508 exif.txt                   0 url_facebook-id.txt
    0 find.txt                   6684 url_histogram.txt
    0 find_histogram.txt          0 url_microsoft-live.txt
    0 gps.txt                     12 url_searches.txt
    0 hex.txt                     156 url_services.txt
   32 ip.txt                      0 vcard.txt
    4 ip_histogram.txt           16432 windirs.txt
   12 jpeg/                      20800 winpe.txt
   504 jpeg.txt                  1864 winprefetch.txt

```















































 jpeg		 kml	0 KB
 aes_keys	1 KB	 lightgrep	0 KB
 alerts	0 KB	 lightgrep_histogram	0 KB
 ccn	3 KB	 packets.pcap	194 KB
 ccn_histogram	1 KB	 rar	1 KB
 ccn_track2	0 KB	 report	105 KB
 ccn_track2_histogram	0 KB	 rfc822	3,727 KB
 domain	23,026 KB	 tcp	19 KB
 domain_histogram	189 KB	 tcp_histogram	1 KB
 elf	0 KB	 telephone	57 KB
 email	1,695 KB	 telephone_histogram	5 KB
 email_histogram	36 KB	 url	70,106 KB
 ether	24 KB	 url_facebook-address	1 KB
 ether_histogram	1 KB	 url_facebook-id	0 KB
 exif	506 KB	 url_histogram	6,682 KB
 find	0 KB	 url_microsoft-live	0 KB
 find_histogram	0 KB	 url_searches	9 KB
 gps	0 KB	 url_services	155 KB
 hex	0 KB	 vcard	0 KB
 ip	32 KB	 windirs	16,429 KB
 ip_histogram	1 KB	 winpe	20,799 KB
 jpeg	503 KB	 winprefetch	1,864 KB
 json	1,895 KB	 zip	29,624 KB

Figure 17: Screenshot from Windows Explorer of the Output Directory Created by the *bulk_extractor* run

1896 json.txt

29624 zip.txt

Many of the feature files and histograms are populated with data. Additionally, there were some JPEG files carved and placed in the *jpeg* directory. In the following sections, we demonstrate how to look at these results to discover more information about the disk user and the files contained on the disk image.

9.2 Digital Media Triage

Digital media triage is the process of using the results of a rapid and automated analysis of the media, performed when the media is first encountered to determine if the media is likely to have information of intelligence value and, therefore, should be prioritized for immediate analysis. *bulk_extractor* performs bulk data analysis to help investigators quickly decide which piece of digital media is the most relevant and useful to an investigation. Thus, *bulk_extractor* can be used to aid in investigations (through the identification of new leads and social networks) rather than just aiding in conviction-support (through the identification of illegal materials)[?].

In this example, we look at the `charlie-2009-12-11.E01` image to quickly assess what kinds of information useful to an investigation might be present on the disk. For the purposes of this example, we will assume we are investigating corporate fraud and trying to discover the answers to the following questions:

- Who are the users of the drive?
- Who is this person communicating with?
- What kinds of websites have they have been visiting most often?
- What search terms are used?

To answer many of these questions, we look at the identify information on the drive including email addresses, credit card information, search terms, Facebook IDs, domain names and vCard data. The output files created by *bulk_extractor* contain all of this type of information that was found on the disk image.

The scenario setup leads us to believe that Charlie is the user of the this drive (based on the name of the disk image). First, we look at `email.txt` to find information about the email addresses contained on the disk. The first two lines of the email features found are the following (each block of text represents one long line of offset, feature and context):

50395384	n\x00o\x00m\x00b\x00r\x00e\x00_\x001\x002\x003\x00@\x00h\x00o\x00t
	\x00m\x00a\x00i\x00l\x00.\x00c\x00o\x00m\x00e\x00m\x00p\x00l\x00o\x00\x00\x0A\x00
	\x09\x00n\x00o\x00m\x00b\x00r\x00e\x00_\x001\x002\x003\x00@\x00h\x00o\x00t\x00m
	\x00a\x00i\x00l\x00.\x00c\x00o\x00m\x00\x0A\x00\x09\x00m\x00i\x00n\x00o\x00m\x00b\x00
50395432	m\x00i\x00n\x00o\x00m\x00b\x00r\x00e\x00@\x00m\x00s\x00n\x00.\x00c
	\x00o\x00m\x00i\x00l\x00.\x00c\x00o\x00m\x00\x0A\x00\x09\x00m\x00i\x00n\x00o\x00m
	\x00b \x00r\x00e\x00@\x00m\x00s\x00n\x00.\x00c\x00o\x00m\x00\x0A\x00\x09\x00e\x00j
	\x00e\x00m\x00p\x00l\x00

It is important to note that UTF-16 formatted text is escaped with `\x00`. This means that `"\x00t \x00e \x00x \x00t"` translates to `"text."` The first two features found are `"nombre_123@hotmail.com"` and `"minombre@msn.com."` Both of the offset values, 50395384 and 50395432, are early on the disk. At this point, there is no way to know

if either of these email addresses are of any significance unless they happen to belong to a suspect or person related to the investigation. The first set of email features found appear on the disk printed in UTF-16 formatted text, like the lines above.

Further down in the feature file, we find the following:

```
9263459 charlie@m57.biz 21)(88=Charlie <charlie@m57.biz>)(89\x0D\x0A      =Pat
9263497 pat@m57.biz          =Pat McGoo <pat@m57.biz>)(8B=WELCOME TO
```

Finding Charlie's email address on the computer begins to further confirm the assumption that this is his computer. The `email_histogram.txt` file provides important information. It shows the most frequently occurring email addresses found on the disk. The following is an excerpt from that top of that file:

```
n=875    mozilla@kewis.ch          (utf16=3)
n=651    charlie@m57.biz (utf16=120)
n=605    ajbanck@planet.nl
n=411    mikep@oeone.com
n=395    belhaire@ief.u-psud.fr
n=379    premium-server@thawte.com      (utf16=11)
n=356    lilmatt@mozilla.com
n=312    cedric.corazza@wanadoo.fr
```

This histogram output shows us that Charlie's email address is the second most frequently occurring name on the disk. It would likely be the first but, as described in the scenario description, this company has only been in business for three weeks and its employees are new users of the computers. Looking at this histogram file also gives us some insight into who the user of this disk is communicating with. Those email addresses occurring most frequently that are not part of the software installed on the machine (such as `ajbanck@planet.nl`) might indicate addresses of people with whom the drive user is corresponding or they may result from other software or web pages that were downloaded. (In this case, the email is from a Firefox extension.)

The file `domain.txt` provides a list of all the "domains" and host names that were found. The sources include URLs, email and dotted quads. Much of the beginning of the feature file is populated with `microsoft.com` domains. This is largely due to the fact that the disk image is from a Windows machine. Further down in the file we find the following:

```
53878576    www.uspto.gov    <a href="http://www.uspto.gov/patft/index.htm
53879083    www.uspto.gov    <A HREF="http://www.uspto.gov/patft/help/help
53880076    ebiz1.uspto.gov  <A HREF="http://ebiz1.uspto.gov/vision-service/
53880536    ebiz1.uspto.gov  <A HREF="http://ebiz1.uspto.gov/vision-service/
```

The domains that were found make sense given that the disk image was obtained from a startup company that deals with patents. Many of the domains found in the file are also in UTF-16 format (with "escaped" characters). It is also worth noting as users browse the domain output file that domains are common in compressed data.

The `domain_histogram.txt` file provides a histogram of the domains found on the disk image. It tends to give us better information for digital media triage than the `domain.txt` file as it provides information about which domains most frequently appear on the disk image and not just the order in which they were found. The beginning of the histogram file looks like the following:

```
n=10749 www.w3.org
n=6670 chroniclingamerica.loc.gov
n=6384 openoffice.org
n=5998 www.uspto.gov
n=5733 www.mozilla.org
n=5212 www.osti.gov
n=4952 www.microsoft.com
n=4470 patft.uspto.gov
```

Many of these domains are part of the operating system, such as openoffice.org, but some are not, such as www.uspto.gov. The histogram file provides insight into the users activity on the machine and which sites they were most frequently visiting.

The file rfc822.txt primarily provides email headers and HTTP headers both of which are in a format specified by RFC822, the Internet Message Standard. It can be useful to see the subject of emails that have been sent and information from HTTP requests. The following is an excerpt from the text file:

```
114074196 SUBJECT:softabs ll|micro)\x5CW?cap\x00SUBJECT:softabs\x00SUBJECT:Caili
114074212 SUBJECT:Cailis SUBJECT:softabs\x00SUBJECT:Cailis\x00\x00SUBJECT:st0ck
114074228 SUBJECT:st0ck SUBJECT:Cailis\x00\x00SUBJECT:st0ck\x00\x00\x00SUBJECT:Your
114074244 SUBJECT:Your Personal Quarantine Folder
SUBJECT:st0ck\x00\x00\x00SUBJECT:Your Personal Quarantine Folder\x00SUBJECT:rolex\x00
114074284 SUBJECT:rolex arantine Folder\x00SUBJECT:rolex\x00\x00\x00SUBJECT:(bro
```

Much of what is found in the file shown above are spam messages.

Telephone numbers found on the disk image are stored in telephone.txt. This following numbers found in the file are clearly for technical support (found within installed software):

```
88850883 (800) 563-9048 rmation centre: (800) 563-9048\x0D\x0A<BR><b><i>Tech
88850995 (905) 568-4494 indows&nbsp;95: (905) 568-4494\x0D\x0A<BR> Microsoft
88851056 (905) 568-2294 ice components: (905) 568-2294\x0D\x0A<BR> Other sta
88851111 (905) 568-3503 hnical support: (905) 568-3503\x0D\x0A<BR> Priority
88851162 (800) 668-7975 rt information: (800) 668-7975\x0D\x0A<BR> Text Tele
```

The next set of "telephone" numbers are clearly bogus numbers:

```
3649684174 008-017-0108 WA,98366,1,4031-008-017-0108,City of Port Or
3649684741 000-031-0009 98337,0.13,3768-000-031-0009,Kitsap County C
3649818237 000-001-0005 8312,2.25,"3768-000-001-0005, 3768-000-003-0
3649818274 000-004-0002 0-003-003, 3768-000-004-0002, 3768-000-005-0
```

Finally, many of the numbers found are legitimate ones. These numbers were all found in GZIP compressed data:

```
3772517888-GZIP-28322 (831) 373-5555 onterey - <nobr>(831) 373-5555</nobr><br><a cl
3772517888-GZIP-29518 (831) 899-8300 Seaside - <nobr>(831) 899-8300</nobr><br><a cl
3772517888-GZIP-31176 (831) 899-8300 Seaside - <nobr>(831) 899-8300</nobr><br><a cl
```

Typically, the file telephone_histogram.txt is the best place to look for phone numbers. In this file, the non-digits are extracted from the phone numbers. The following is an excerpt from the beginning of that file:

```
n=42 +14159618830
n=35 8477180400
n=24 +27112570000
n=24 2225552222
```

n=18	8005043248
n=15	2225551111
n=13	8662347350
n=12	8772768437
n=11	2522277013

Investigators looking for specific information about the user of a disk image or who they have been communicating with can look quickly at this file and see how frequently numbers appear. It also consolidates the numbers in a way that makes it easy for investigators looking for a specific number or set of numbers to see them quickly.

Finally, in performing digital media triage on the disk image, investigators would like to know what specific URLs have been visited and what search terms the user has been using. The set of URL files provided as output provide insight into this information. First, `url.txt` contains the URLs found on the disk. The following is an excerpt from that file (note that the UTF-16 formatted information is escaped):

```
175165385 http://www.unicode.org/reports/tr25/#_TocDelimiters E and U+23DF:\x0A#
http://www.unicode.org/reports/tr25/#_TocDelimiters\x0A\x5Cu23DE = \x5CuE13B

159045397 h\x00t\x00t\x00p\x00:\x00/\x00/\x00w\x00w\x00w\x00.\x00d\x00o\x00w
\x00n\x001\x00o\x00a\x00d\x00.\x00w\x00i\x00n\x00d\x00o\x00w\x00s\x00u\x00p
\x00d\x00a\x00t\x00e\x00.\x00c\x00o\x00m\x00/\x00m\x00s\x00d\x00o\x00w\x00n\x001\x00o
\x00a\x00d\x00/\x00u\x00p\x00d\x00a\x00t\x00e\x00/\x00s\x00o\x00f\x00t\x00w\x00a\x00r
\x00e\x00/\x00s\x00e\x00c\x00u\x00/\x002\x000\x000\x008\x00/\x000\x006\x00/\x00w\x00i
\x00n\x00d\x00o\x00w\x00s\x00x\x00p\x00-\x00k\x00b\x009\x005\x001\x003\x007\x006\x00-
\x00v\x002\x00-\x00x\x008\x006\x00-\x00e\x00n\x00u\x00_\x00e\x009\x00b\x006\x008\x00c
\x005\x00e\x006\x003\x00a\x00c\x00b\x005\x007\x008\x006\x00a\x000\x005\x00b\x005\x003
\x00b\x004\x00 \xB4\xF4\x82\x94C\xE3\xB6C\xB1p\x9Ae\xBC\x82,wh\x00t\x00t\x00p\x00:
\x00/\x00/\x00w\x00w\x00w\x00.\x00d\x00o\x00w\x00n\x001\x00o\x00a\x00d\x00.\x00w
\x00i\x00n\x00d\x00o\x00w\x00s\x00u\x00p\x00d\x00a\x00t\x00e\x00.\x00c\x00o
\x00m\x00/\x00m\x00s\x00d\x00o\x00w\x00n\x001\x00o\x00a\x00d\x00/\x00u\x00p\x00d
\x00a\x00t\x00e\x00/\x00s\x00o\x00f\x00t\x00w\x00a\x00r\x00e\x00/\x00s\x00e\x00c\x00u
\x00/\x002\x000\x000\x008\x00/\x000\x006\x00/\x00w\x00i\x00n\x00d\x00o\x00w\x00s\x00x
\x00p\x00-\x00k\x00b\x009\x005\x001\x003\x007\x006\x00-\x00v\x002\x00-\x00x\x008\x006
\x00-\x00e\x00n\x00u\x00_\x00e\x009\x00b\x006\x008\x00c\x005\x00e\x006\x003\x00a\x00c
\x00b\x005\x007\x008\x006\x00a\x000\x005\x00b\x005\x003\x00b\x004\x003\x003\x002\x004
\x006\x005\x00d\x00e\x00

175197993 http://www.uspto.gov/patft/index.html enter>\x0A<a href="http://www.
uspto.gov/patft/index.html"><IMG BORDER="0
```

The file `url_histogram.txt` provides the histogram of the potential urls. In that file, UTF-16 formatted text is converted to UTF-8. Note that not all URLs contained in the histogram file are accurate. The are actually URLs that were typed into a web browser. The following are lines taken from that file:

n=3922	http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul	(utf16=2609)
n=859	http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xu	(utf16=858)
...		
n=2	http://math.nist.gov/~KRemington/papers/europvm.ps	
n=2	http://math.nist.gov/~MDonahue/pubs/nan.ps.gz	
n=2	http://math.nist.gov/~RBoisvert/publications/ADL95.ps.gz	
n=2	http://math.nist.gov/~RBoisvert/publications/IMACS97.ps.gz	

Because the histogram file converts the UT-16 formatted text to UTF-8, the histogram file is more human readable than the `url.txt` file alone. The files `url_facebook.txt`, `url_microsoft-live`, `url_services` and `url_searches` all extract specific types of information from URLs. The most useful for digital media triage is likely the file `url_searches.txt` because it shows histogram of searches from the disk image. Searches frequently convey intent. The following is an excerpt from that file:

```
n=60      1
n=53      exotic+car+dealer
n=41      ford+car+dealer
n=34      2009+Shelby
n=25      steganography
n=23      General+Electric
n=23      time+travel
n=19      steganography+tool+free
n=19      vacation+packages
n=16      firefox
n=16      quicktime
n=14      7zip
```

The file `ccn.txt` provides credit card numbers that have been found on the disk. Based on the scenario set-up for this disk image, credit card numbers are not necessarily highly relevant to this investigation. However, *bulk_extractor* did find some credit card numbers on this disk image that are not actually credit card numbers; This is common behavior so it is worth examining the file here to demonstrate how it can be used in other investigations. The credit card number finder considers a pattern of digits and uses the Luhn checksum algorithm and the distribution of digits and the local context to identify potential credit card numbers. It is important to note that there are frequently false positives. The first few lines of the `ccn.txt` file for this disk image look like the following:

```
88284672 -GZIP -177427 5273347458642687 734B55CD5\x0A5273347458642687\x0AC0841BAFA1B4C28
4814857216 -GZIP -793 4015751530102097 eb0.d=0;eb0.rnd=4015751530102097;eb0.title="";eb
4909069775 6543210123456788 \x0Addadd7540 add '6543210123456788' 0.499999999
4909069811 6543210123456788 4999999999 -> '6543210123456788' Inexact Rounde
4909069861 6543210123456788 \x0Addadd7541 add '6543210123456788' 0.5
4909069897 6543210123456788 5 -> '6543210123456788' Inexact Rounde
4909069947 6543210123456788 \x0Addadd7542 add '6543210123456788' 0.500000001
5304221350 5678901234560000 +4 -> 5678901234560000\x0D\x0Addshi052 shift
5612375618 6543210123456788 \x0D\x0Aaddx6240 add '6543210123456788' 0.499999999
5612375654 6543210123456788 4999999999 -> '6543210123456788' Inexact Rounde
5612375703 6543210123456788 \x0D\x0Aaddx6241 add '6543210123456788' 0.5
5612375739 6543210123456788 5 -> '6543210123456788' Inexact Rounde
5612375788 6543210123456788 \x0D\x0Aaddx6242 add '6543210123456788' 0.500000001
5612715901 5700122152274696 div4036 divide 5700122152274696 5700122152251
```

In the above example, '525273347458642687' looks like it could be a valid credit card number from the context (`\x0A` is a new line). The number '4015751530102097' looks like a random number in a piece of Java Script. Note that both of those numbers were compressed; the offset indicates they were found in GZIP streams (shown as a number followed by '-GZIP'). The numbers whose context include "Inexact Rounde" are actually from Python source code and not credit card numbers at all. Again, the `ccn.txt` tends to alert on a lot of false positives.

The `ccn_track2.txt` file did not find any information in this disk image but is also useful for credit card fraud and identity theft investigations. It will contain credit card

track 2 information found on the disk image.

Using the files produced by *bulk_extractor* described above, an investigator can quickly review a disk image for important information that is relevant to an investigation and find actionable intelligence quickly.

9.3 Analyzing Imagery

The scenario described in the M57 Patents data is not necessarily relevant to an imagery investigation. However, there is imagery information on the disk. We use that information here to demonstrate how imagery information can be analyzed by an investigator using *bulk_extractor*.

The file in the output directory, `jpeg.txt`, lists all JPEGs that were found on the disk whether they were carved or not. *bulk_extractor* was run with default values meaning that only encoded JPEGs were carved. The following excerpt from the JPEG file shows information about JPEGs found on the disk image:

```
54798824      ../Output/charlie-2009-12-11/jpeg/54783488.jpg <fileobject><filename>
../Output/charlie-2009-12-11/jpeg/54783488.jpg</filename><filesize>15336</filesize>
<hashdigest type='md5 '>13823ce7c21587d31f6eb4474612e660
</hashdigest></fileobject>
```

The JPEG described above was not carved because it was not encoded. However, the first section “../Output/charlie-2009-12-11/jpeg/54783488.jpg” shows where the file would be found in the output directories if it had been carved. The next section of information also gives the file size, the hash type (in this case ‘md5’) and the hash value of the file (in this case 13823ce7c21587d31f6eb4474612e660). Note that this may not match the hash value of the file in the original file system as *bulk_extractor* cannot properly carve fragmented files.

Information about encoded JPEGs can also be found in the `jpeg.txt` file. The following excerpt shows a description of a JPEG found in a GZIP format on the disk:

```
3771686400-GZIP-8332      ../Output/charlie-2009-12-11/jpeg/3771686400-GZIP-0.jpg
<fileobject><filename>../Output/charlie-2009-12-11/jpeg/3771686400-GZIP-0.jpg
</filename><filesize>8332</filesize><hashdigest type='md5 '>
5b77035c983b04996774370f735ea72a</hashdigest></fileobject>
```

The JPEG described above was carved and can be found in the `/jpeg` output directory in the file named `3771686400-GZIP-0.jpg`. The file also gives information about the filesize, hash type and hash ID. That file is shown in the directory output shown below along with all of the encoded JPEGs that were found on the disk image and were carved. The contents of the `/jpeg` directory are as follows:

```
10037939712-GZIP-0.jpg  5324841013-ZIP-0.jpg
10117679783-ZIP-0.jpg  6039195136-GZIP-0.jpg
3761630720-GZIP-0.jpg  6039215616-GZIP-0.jpg
3764534784-GZIP-0.jpg  6039223808-GZIP-0.jpg
3771686400-GZIP-0.jpg  6039232000-GZIP-0.jpg
3771706880-GZIP-0.jpg  6039244288-GZIP-0.jpg
3771715072-GZIP-0.jpg  6039301632-GZIP-0.jpg
3771723264-GZIP-0.jpg  6039318016-GZIP-0.jpg
3771735552-GZIP-0.jpg  6883925636-ZIP-0.jpg
3771792896-GZIP-0.jpg  6884040324-ZIP-0.jpg
3771809280-GZIP-0.jpg  6884056948-ZIP-0.jpg
```



Figure 18: A JPEG carved from encoded data on the M57 Patents disk image

```
3771833856-GZIP-0.jpg    7276064256-GZIP-0.jpg
3771858432-GZIP-0.jpg    7279128576-GZIP-0.jpg
429788672-GZIP-0.jpg     8877243047-ZIP-0.jpg
5310405287-ZIP-0.jpg     9948655104-GZIP-0.jpg
```

All of these JPEG files can be viewed and used by investigators. The filename is the forensic path of where the JPEG was found. The file 3771686400-GZIP-0.jpg mentioned above is shown in Figure 18.

9.4 Password Cracking

The *wordlist* generates a list of all the words found on the disk that are between 6 and 14 characters long. The word list that is generated by the scanner can be very useful in determining combinations of words to use for password cracking. The scanner is enabled by default because it slows down the *bulk_extractor* run significantly. To show the word list in this example, *bulk_extractor* was run again on the M57 Patents scenario data with the *wordlist* scanner enabled. Running *bulk_extractor* on the command line with it enabled produces the following output:

```
C:\be>bulk_extractor -e wordlist -o ../Output/charlie-wordlist charlie-2009-12-11.E01
```

```
bulk_extractor version: 1.4.0
Input file: charlie-2009-12-11.E01
Output directory: ../Output/charlie-wordlist
Disk Size: 10239860736
Threads: 4
12:58:46 Offset 67MB (0.66%) Done in 1:14:55 at 14:13:41
...
14:03:24 Offset 10217MB (99.78%) Done in 0:00:08 at 14:03:32
All data are read; waiting for threads to finish...
Time elapsed waiting for 4 threads to finish:
    (timeout in 60 min .)
Time elapsed waiting for 4 threads to finish:
    8 sec (timeout in 59 min 52 sec.)
Thread 0: Processing 10200547328
Thread 1: Processing 10234101760
Thread 2: Processing 10183770112
Thread 3: Processing 10217324544

Time elapsed waiting for 1 thread to finish:
    14 sec (timeout in 59 min 46 sec.)
Thread 3: Processing 10217324544
```

```
All Threads Finished!
Producer time spent waiting: 3627.92 sec.
Average consumer time spent waiting: 4.1518 sec.
*****
** bulk_extractor is probably CPU bound. **
** Run on a computer with more cores **
```

```

**          to get better performance.          **
*****
Phase 2. Shutting down scanners
Phase 3. Uniquifying and recombining wordlist
Phase 3. Creating Histograms
  ccn histogram...   ccn_track2 histogram...   domain histogram...
  email histogram... ether histogram...   find histogram...
  ip histogram...   lightgrep histogram...   tcp histogram...
  telephone histogram... url histogram...   url microsoft-live...
  url services...   url facebook-address... url facebook-id
  url searches...Elapsed time: 4065.09 sec.
Overall performance: 2.51898 MBytes/sec
Total email features found: 152775

```

Note that it took 3991.71 seconds to run *bulk_extractor* without the *wordlist* scanner enabled and, in this case, it took 4065.09 seconds with *wordlist* enabled. The new output directory contains a file called *wordlist.txt*. That file has both filenames and words in it. The following is an excerpt from that file:

50497556	usemodem.jpg
50497624	usemsn.jpg
50497692	usemsnow.jpg
50497760	welcome.htm
50497828	whereNow.htm
50497896	xmlutil.js
50497987	^Photoshop
50498009	Resolution
50498050	Global
50498057	Lighting
50498090	Global
50498097	Altitude
50498153	Copyright
50498181	Japanese
50498229	Halftone
50498238	Settings
50498335	Transfer

The wordlist contains ALL words found on the disk between 6 and 14 characters long. Automated programs can be used to generate passwords from combinations of these words. The *wordlist* scanner also generates a split wordlist containing the same words found in the *wordlist.txt* file with all words deduplicated, sorted by size and alphabetized. The following is an excerpt from the file *wordlist_split_000.txt* generated from the disk image:

concluded 1
concluder/2
concluder/M
concluir/XQ
conclurai/x
conclusion,
conclusion.
conclusionone
conclusions
conclusive,

The split wordlist is the file that is typically fed to password cracking software.

9.5 Post Processing

The programs **identify_filenames.py** and **bulk_diff.py** can provide further insight into the data contained on the disk image. The **identify_filenames.py** program can be used on the feature files produced from the *bulk_extractor* run to show the file location of the features that were found. Running the program on all of the feature files produced by the *bulk_extractor* run produces the following output (where *charlie-2009-12-11* is the *bulk_extractor* output directory and *charlieAnnotatedOutput* is where all the annotated files are written):

```
C:\be\>identify_filenames.py -all charlie-2009-12-11 charlieAnnotatedOutput
Reading file map by running fiwalk on charlie-2009-12-11.E01
Processed 1000 fileobjects in DFXML file
Processed 2000 fileobjects in DFXML file
...
Processed 39000 fileobjects in DFXML file
Processed 40000 fileobjects in DFXML file
feature_file: aes_keys.txt
feature_file: ccn.txt
feature_file: domain.txt
feature_file: email.txt
feature_file: ether.txt
feature_file: exif.txt
feature_file: ip.txt
feature_file: jpeg.txt
feature_file: json.txt
feature_file: rar.txt
feature_file: rfc822.txt
feature_file: telephone.txt
feature_file: url.txt
feature_file: windirs.txt
feature_file: winpe.txt
feature_file: winprefetch.txt
feature_file: zip.txt
*****
** Total Features:    754038 **
** Total Located:    754038 **
*****
```

Note, in this example that **fiwalk** is installed on the computer running the **identify_filenames.py** program. The directory *charlieAnnotatedOutput* contains all of the annotated feature files, showing the file location of the features. The directory contents are as follows:

annotated_aes_keys.txt	annotated_rar.txt
annotated_ccn.txt	annotated_rfc822.txt
annotated_domain.txt	annotated_telephone.txt
annotated_email.txt	annotated_url.txt
annotated_ether.txt	annotated_windirs.txt
annotated_exif.txt	annotated_winpe.txt
annotated_ip.txt	annotated_winprefetch.txt
annotated_jpeg.txt	annotated_zip.txt
annotated_json.txt	

The annotated files display the feature with the file in which the feature was found (where it was identified by the program). The following is an excerpt from the *annotated_email.txt* file:

27767966 pat@m57.biz	m: "Pat McGoo" <pat@m57.biz>\x0D\x0Ato: <charlie@ Documents
----------------------	---

```

and Settings/Charlie/Application Data/Thunderbird/Profiles/4zy34x9h.default/Mail/Local
Folders/Inbox dcb794e350bd198c4279614eae6c8b76

27767985 charlie@m57.biz @m57.biz>\x0D\x0ATo: <charlie@m57.biz>,\x0D\x0A\x09<jo@m
57.biz Documents and Settings/Charlie/Application Data/Thunderbird/Profiles/4zy34x9h.
default/Mail/Local Folders/Inbox dcb794e350bd198c4279614eae6c8b76

27768022 terry@m57.biz jo@m57.biz>,\x0D\x0A\x09<terry@m57.biz>\x0D\x0AAX-ASG-Orig-
Su Documents and Settings/Charlie/Application Data/Thunderbird/Profiles/4zy34x9h.def
ault/Mail/Local Folders/Inbox dcb794e350bd198c4279614eae6c8b76

```

The email address "pat@m57biz" was found in the file Documents and Settings/Charlie/Application Data/Thunderbird/Profiles/4zy34x9h.default/Mail/Local Folders/Inbox and investigators can refer to that location on the disk image to view the full text.

The program **bulk_diff.py** shows the difference between two *bulk_extractor* runs. In this case, we used a disk image from the same user ("charlie") taken almost a month before the disk image that has been used throughout this example. The disk image we have been using throughout this example is dated December 11, 2009. The older disk image we downloaded for comparison is dated November 17, 2009. The earlier disk image data is stored in a file named **charlie-2009-11-17.E01** and can be downloaded from <http://digitalcorpora.org/corp/nps/scenarios/2009-m57-patents/drives-redacted/>.

After running *bulk_extractor* using the earlier disk image, we ran the program **bulk_diff.py** on the output of that disk image and on the output of the **charlie-2009-12-11.E01** run. To run, we typed the following, piping the output of the program to a file called **bulkdifffoutput.txt**:

```
■ bulk_diff.py /charlie-2009-11-17 /charlie-2009-12-11 > bulkdifffoutput.txt
```

The output shows the features differences on the disk image. The following is an excerpt of that output:

domain_histogram.txt:			
	#in PRE	#in POST	Value
401	4,470	4,069	patft.uspto.gov
181	3,151	2,970	www.wipo.int
295	3,157	2,862	www.google.com
0	2,537	2,537	l.yimg.com

The output specifically shows the differences in the histograms between the two runs across all of the histogram files that were created. The excerpt above shows that "charlie" (the disk user) visited the domain "patft.uspto.gov" frequently between the time the two images were recorder. It was found 4,069 more times in the later disk image than in the one taken earlier. It also shows that the domain "l.yimg.com" was not found on the earlier disk image but was found 2,537 times on the later disk image. The results are sorted by the amount of the difference. This means that features that are most different appear first. This can be very helpful because those features generally give the most insight into the disk users activity over that period of time.

10 NPS DOMEX Users Image

NPS Test Disk Images are a set of disk images that have been created for testing computer forensic tools. These images are free of non-public Personally Identifiable Infor-

mation (PII) and are approved for release to the general public. The NPS-created data in the images is public domain and free of any copyright restriction; the images may contain some copyrighted data that was made available by the copyright holder. These copyrights, where known, are noted in the files themselves[?].

The NPS DOMEX users image is a disk image of a Windows XP SP3 system that has two users, domexuser1 and domexuser2, who communicate with a third user (domexuser3) via IM and email. The data is available for download at <http://digitalcorpora.org/corp/nps/drives/nps-2009-domexusers/>. For this example, we use the file `nps-2009-domexusers.E01` which includes the full system including the Microsoft Windows executables. Running *bulk_extractor* on the command line produces the following output:

```
C:\be\>bulk_extractor -o ../Output/nps-2009-domexusers nps-2009-domexusers.E01
```

```
bulk_extractor version: 1.4.0
Input file: nps-2009-domexusers.E01
Output directory: ../Output/nps-2009-domexusers2
Disk Size: 42949672960
Threads: 4
16:50:53 Offset 67MB (0.16%) Done in 4:23:43 at 21:14:36
16:51:19 Offset 150MB (0.35%) Done in 3:58:37 at 20:49:56
...
16:13:12 Offset 42849MB (99.77%) Done in 0:00:11 at 16:13:23
16:13:13 Offset 42932MB (99.96%) Done in 0:00:01 at 16:13:14
All data are read; waiting for threads to finish...
Time elapsed waiting for 3 threads to finish:
    (timeout in 60 min .)
Time elapsed waiting for 1 thread to finish:
    6 sec (timeout in 59 min 54 sec.)
Thread 0: Processing 42932895744

Time elapsed waiting for 1 thread to finish:
    12 sec (timeout in 59 min 48 sec.)
Thread 0: Processing 42932895744

All Threads Finished!
Producer time spent waiting: 4254.07 sec.
Average consumer time spent waiting: 89.309 sec.
*****
** bulk_extractor is probably CPU bound. **
** Run on a computer with more cores **
** to get better performance. **
*****
Phase 2. Shutting down scanners
Phase 3. Creating Histograms
    ccn histogram...    ccn_track2 histogram...    domain histogram...
    email histogram...    ether histogram...    find histogram...
    ip histogram...    lightgrep histogram...    tcp histogram...
    telephone histogram...    url histogram...    url microsoft-live...
    url services...    url facebook-address...    url facebook-id...
    url searches...Elapsed time: 4846.74 sec.
Overall performance: 8.86156 MBytes/sec
Total email features found: 8774
```

All of the results from the *bulk_extractor* run are stored in the output directory *nps-2009-domex*. The contents of that directory after the run are as follows:

```
1 aes_keys.txt          1 kml.txt
0 alerts.txt            0 lightgrep.txt
1 ccn.txt               0 lightgrep_histogram.txt
```

1 ccn_histogram.txt	4 packets.pcap
0 ccn_track2.txt	1 rar.txt
0 ccn_track2_histogram.txt	424 report.xml
7364 domain.txt	536 rfc822.txt
44 domain_histogram.txt	1 tcp.txt
0 elf.txt	1 tcp_histogram.txt
1528 email.txt	48 telephone.txt
32 email_histogram.txt	4 telephone_histogram.txt
1 ether.txt	51888 url.txt
1 ether_histogram.txt	0 url_facebook-address.txt
152 exif.txt	0 url_facebook-id.txt
0 find.txt	1240 url_histogram.txt
0 find_histogram.txt	0 url_microsoft-live.txt
0 gps.txt	4 url_searches.txt
0 hex.txt	32 url_services.txt
4 ip.txt	0 vcard.txt
1 ip_histogram.txt	15228 windirs.txt
20 jpeg/	26516 winpe.txt
380 jpeg.txt	1312 winprefetch.txt
316 json.txt	1956 zip.txt

For this example, we will focus on the files that are most important to malware investigations and cyber investigations, showing how those files can be interpreted and used by investigators.

10.1 Malware Investigations

In a malware investigation, investigators are looking for information about programmatic intrusions. In this example, we examine all files that provide information about executables, Windows directory entries and information downloaded from web-based applications. We recommend that "-e xor" be enabled for malware investigations.

The file `windirs.txt` provides information about FAT32 and NTFS directories. It contains most of the disk entries. The following is an excerpt showing one line from the file:

```
281954816      A0001801.dll      <fileobject
src='mft'><atime>2008-10-21T00:45:51Z</atime><attr_flags>8224</attr_flags>
<ctime>2008-10-21T00:45:51Z</ctime><ctime>2008-10-21T00:45:51Z</ctime>
<filename>A0001801.dll</filename><filesize>1000000000000</filesize><filesize_alloc>
0</filesize_alloc><lsn>123437339</lsn><mtime>2008-10-21T00:45:51Z</mtime>
<nlink>1</nlink><par_ref>12017</par_ref><par_seq>3</par_seq><seq>1</seq>
</fileobject>
```

The line from the file gives information about the disk entry `A0001801.dll`. It provides some data about the file including the file size, file creation time (ctime) and time of last file modification (mtime). It is important to note that the error rate for FAT32 entries is high and those entries should be ignored if the drive is not FAT.

For investigations on Windows disk images, such as the `nps-2009-domexusers`, the file `winpe.txt` shows Windows executables related to the Windows Preinstallation Environment. These file entries contain very long lines. The following is **one** line from the file:

```
42753536 87d84154e7789013878c6340a4d2d445 <PE><FileHeader Machine=
"IMAGE_FILE_MACHINE_I386"NumberOfSections="3" TimeDateStamp="1208131815"
PointerToSymbolTable="0"NumberOfSymbols="0"SizeOfOptionalHeader="224">
```



```

<Characteristics><IMAGE_FILE_EXECUTABLE_IMAGE />
<IMAGE_FILE_LINE_NUMS_STRIPPED /><IMAGE_FILE_LOCAL_SYMS_STRIPPED />
<IMAGE_FILE_32BIT_MACHINE/><IMAGE_FILE_DLL /></Characteristics>
</FileHeader><OptionalHeaderStandard Magic="PE32" MajorLinkerVersion="7"
MinorLinkerVersion="10" SizeOfCode="512" SizeOfInitializedData="1536"
SizeOfUninitializedData="0" AddressOfEntryPoint="0x1046" BaseOfCode=
"0x1000" /><OptionalHeaderWindows ImageBase="0x6c6c0000" SectionAlignment
="1000" FileAlignment="200"MajorOperatingSystemVersion="5"
MinorOperatingSystemVersion="1" MajorImageVersion="5"
MinorImageVersion="1" MajorSubsystemVersion="4" MinorSubsystemVersion="0"
Win32VersionValue="0" SizeOfImage="4000" SizeOfHeaders="400" CheckSum="
0x7485" SubSystem="" SizeOfStackReserve="40000"SizeOfStackCommit="1000"
SizeOfHeapReserve="100000" SizeOfHeapCommit="1000" LoaderFlags="0"
NumberOfRvaAndSizes="10"><DllCharacteristics>
<IMAGE_DLL_CHARACTERISTICS_NO_SEH /></DllCharacteristics>
</OptionalHeaderWindows><Sections><SectionHeader Name=".text" VirtualSize
="be" VirtualAddress="1000" SizeOfRawData="200" PointerToRawData="400"
PointerToRelocations="0" PointerToLinenumbers="0" ><Characteristics>
<IMAGE_SCN_CNT_CODE /><IMAGE_SCN_MEM_EXECUTE />
<IMAGE_SCN_MEM_READ /></Characteristics></SectionHeader><SectionHeader
Name=".rsrc" VirtualSize="400" VirtualAddress="2000" SizeOfRawData="400"
PointerToRawData="600" PointerToRelocations="0" PointerToLinenumbers="0"
><Characteristics><IMAGE_SCN_CNT_INITIALIZED_DATA />
<IMAGE_SCN_MEM_READ /></Characteristics></SectionHeader>
<SectionHeader Name=".reloc" VirtualSize="8" VirtualAddress="3000"
SizeOfRawData="200" PointerToRawData="a00" PointerToRelocations="0"
PointerToLinenumbers="0" ><Characteristics><IMAGE_SCN_CNT_INITIALIZED_DATA />
<IMAGE_SCN_MEM_DISCARDABLE /><IMAGE_SCN_MEM_READ /></Characteristics>
</SectionHeader></Sections></PE>

```

The first number is the offset and tells you where to find the file. Most executables are not fragmented. The second is the MD5 hash of the first 4k of the file that can be used to deduplicate and look up the file in the hash database. Finally, the bulk of the information is contained in the <PE> XML block that breaks out all of the Windows PE header information. It contains information about the File header, the characteristics of the file, Windows header information and section header information.

The file `winprefetch.txt` contains the information from carved files Windows Prefetch that were discovered anywhere on the drive. *bulk_extractor* will carve the Prefetch files from unallocated space. This is extremely useful because Prefetch files are frequently deleted. A single line in the prefetch output file is also very long. The following is only the beginning of one line from the file:

```

55758336      MSIEXEC.EXE      <prefetch><os>Windows
XP</os><filename>MSIEXEC.EXE</filename><header_size>152</header_size>
<atime>2008-10-30T03:17:27Z</atime><runs>14</runs><filenames>
<file>\x5CDEVICE\x5CHARDDISKVOLUME1\x5CWINDOWS\x5CSYSTEM32\x5CNTDLL.DLL
</file><file>\x5CDEVICE\x5CHARDDISKVOLUME1\x5CWINDOWS\x5CSYSTEM32\x5CKERNEL32.DLL
...

```

Printing the line out here would cover almost two pages. It includes a lot of information about the Prefetch file including the name of the executable, the name of the DLLs, the directory of DLLs, the atime, the number of runs, the serial number, and the ctime. The Prefetch file is searchable and useable by investigators searching for EXEs or DLLs related to a malware investigation.

JSON is the JavaScript Object Notation (used in Facebook, etc). The file `json.txt`

provides the offset, JSON and MD5 hash of the JSON information found on the disk image. *bulk_extractor* is great at finding JSON in compressed streams and HIBER files. The following are a few lines from the JSON file:

```
62836579 {"ask":["Ask"],"delicious":["Del.icio.us"],"digg":["Digg"],"email":["Email"],
"favorites":["Favorites"],"facebook":["Facebook"],"fark":["Fark"],"furl":["Furl"],
"google":["Google"],"live":["Live"],"myspace":["MySpace"],"myweb":["Yahoo MyWeb",
"yahoo-myweb"],"newsvine":["Newsvine"],"reddit":["Reddit"],"sk*rt":["Sk*rt","skrt"],
"slashdot":["Slashdot"],"stumbleupon":["StumbleUpon"],"su":["su"],"stylehive":["Stylehive"],
"tailrank":["Tailrank","tailrank2"],"technorati":["Technorati"],"thisnext":
["ThisNext"],"twitter":["Twitter"],"ballhype":["BallHype"],"yardbarker":
["Yardbarker"],"kaboodle":["Kaboodle"],"more":["More ..."]}
26d3b8c5010f4d39250dab3a1c1b839e

62842797 ["6jb4","3j1d","v1me","gu83","uefc","fq1j","r5l7","ftho","gdq9","717h",
"24b7","d0en","ads7","m9b4","n0lq","42c3","p5mp","7hbi","f0g6","7v98","mv86",
"d0ns","9a8a","64gg","jogl","cehp","mu2r","6h7h","sntb","94ds","n1fv","3a2i",
"3end","l42s","a9j","q3dj","s150","di3s","3nu5","sk74","e39d","mkvj","482d","kfej",
"nlcv","eroi","m6ee","rvaa","9nis","ef6b","g00q","b4hp","kbpq","bm4l","f7iu",
"e5gb","1sbj","rk0a","ck86","1etp","26sr","fivt","3v95","foqq","vtmj","canb",
"bchv","ku35","q4p9","gdk","gng8","mdb9","ejjg","27k9","30mf","nene",
"smmm","q204","83ot","6kbr","df1o","lq0j","nh32","ebso","d6t5","f2dp",
"3sqp","i4cs","6k7b","a1pv","ki2l","1f7","d6lv","u7r5","9t0e","5h0l","j8kn",
"7akj","9tj","jmu3","1ir1"] 5a04af7518ad74c497c9e74b7025736e

64044544-GZIP-610 ["Top","Left","Right","Bottom"] 5354ef6838974b1979e49ee379883c56
```

Some of the JSON features found, such as the one located at '62836579', are comprised of a lot of information in the notation. Other JSON features are very short, such as the feature located at in the GZIP compressed stream at '64044544-GZIP-610.' All of the lines contain the MD5 hash of the JSON that is used for deduplication.

The file `elf.txt` typically contains information about ELF executables, which is the executable file format for Linux and Android systems. The sample corpus used in this example is from a Windows machine and does not contain any ELF executables.

10.2 Cyber Investigations

Cyber investigations cover a wide variety of areas. However, most involve looking for encryption keys, hash values or information about ethernet packets. *bulk_extractor* finds all of those things on the disk and writes them to different output files. Of note, *bulk_extractor* also finds information in Base64 encoding and decompresses fragments of Windows Hibernation files. There are not specific files created for that processing; the information found in data with these encodings will be processed by other scanners and stored in the appropriate feature files. The fact that a feature came from encoded data will be indicated in the forensic path. The information contained therein may very well be relevant to cyber investigations.

AES encryption implementation system sometimes leaves keys in memory and *bulk_extractor* finds those keys, usually in RAM, Swap or hibernation files. The keys can sometimes be used to decrypt AES encrypted material. The file `aes.txt` contains the keys that are found. There was only one AES key found on the `nps-2009-domexusers` disk image. The following is the line that describes it from the keys file including the offset, key and key size descriptor (AES256):

```
1608580652      28 90 90 5e f7 ce b4 a7 2b 7d d9 45 d8 b0 56 99 97 f4 42
33 35 f1 54 9a 79 36 e7 1c 94 02 28 78  AES256
```

The file `hex.txt` contains extracted hexadecimal strings of a special length. The block sizes contained within it are either 128 or 256 due to the fact that those are the sizes used for encryption keys and hash values. The disk image used in this example does not have any of those and the file is blank.

`bulk_extractor` produces network information including PCAP files, Ethernet addresses, and TCP/IP connections. The files `ether.txt` and `ether_histogram.txt` provide a list of ethernet addresses from packets and ASCII. These are the addresses found on the disk and located in `ether.txt`:

```
2435863552      00:0C:29:26:BB:CD      (ether_dhost)
2435863552      00:50:56:E0:FE:24      (ether_shost)
2435865088      00:0C:29:26:BB:CD      (ether_dhost)
2435865088      00:50:56:E0:FE:24      (ether_shost)
22637986225     00:80:C7:8F:6C:96      apter.\x0AExample: 00:80:C7:8F:6C:96\x00\x00
```

The file `ether_histogram.txt` groups these ethernet addresses in a histogram:

```
n=2      00:0C:29:26:BB:CD
n=2      00:50:56:E0:FE:24
n=1      00:80:C7:8F:6C:96
```

Packets likely traveled from 00:0C:29:26:BB:CD to 00:50:56:E0:FE:24. The other usage has Ethernet addresses in UTF-16 format.

The file `ip.txt` contains IP addresses from packet carving, not from dotted quads. The following is an excerpt from that file:

```
2435865102      inet_ntop win32 struct ip L (src) cksum-ok
2435865102      inet_ntop win32 struct ip R (dst) cksum-ok
2805534669      123.12.0.192      sockaddr_in
8694397397      135.5.0.234      sockaddr_in
9047318477      123.12.0.192      sockaddr_in
9446959573      135.5.0.234      sockaddr_in
11295228937     1.70.0.1          sockaddr_in
```

The *L* or *R* in the 'struct ip' information indicates Local or Remote. This line also includes the IP checksum is ok. The value could also be listed as "cksum-bad" to indicate it is bad. Bad checksums may indicate a false positive and not a legitimate IP address. Finally, the "sockaddr_in" indicates the IP address is from a "sockaddr_in" structure. The file `ip_histogram.txt` removes the random noise that is found in the `ip.txt`. Here is an excerpt from the histogram file:

```
n=5      2.172.0.101
n=4      123.12.0.192
n=4      inet_ntop win32
n=3      135.5.0.234
n=2      209.85.147.109
n=2      65.55.15.242
```

The file `packets.pcap` is a pcap file made from carved packet. To view that file, use any packet analysis tool you like (such as `tcpdump`). Only packets carved from a PCAP file will have the correct packet time stamp; others will given a time in 1970.

Finally, the file `tcp.txt` contains details about TCP (and UDP) network flows. It contains more detail than `ip.txt` but investigators should be careful of false positives, as there are often many in this file. The following are the two lines found in that file:

2435863566	inet_ntop win32:80 -> inet_ntop win32:1034 (TCP)	Size: 1472
2435865102	inet_ntop win32:80 -> inet_ntop win32:1034 (TCP)	Size: 1252

The file `tcp_histogram.txt` often provides further insight into the tcp information found on the disk image. In this case, it does not because there were only two features found. It is important to note that the histogram file still contains a lot of false positives.

11 Troubleshooting

Every forensic tool crashes at times because the tools are routinely used with data fragments, non-standard codings, etc. One major issue is that the evidence that makes the tool crash typically cannot be shared with the developer. The *bulk_extractor* system implements checkpointing to protect the user and the results. *bulk_extractor* checkpoints the current page in the file `report.xml`. After a crash, the user can just hit the up-arrow at the command line prompt and return. *bulk_extractor* will restart at the next page.

All *bulk_extractor* users should join the *bulk_extractor* users Google group for more information and help with any issues encountered. To join, send an email to bulk_extractor-users+subscribe@googlegroups.com.

For the most part, the only kind of debugging *bulk_extractor* users should be doing is turning off scanners. If *bulk_extractor* crashes repeatedly on a data set, the scanners can all be disabled and then turned back on, one by one, until it crashes again. Then, the user can report the specific scanner that made *bulk_extractor* crash on their disk image. In general, users who experience crashes should feel free to report issues and problems to the developers via the Google users group.

Users running the 32-bit version of *bulk_extractor* may occasionally encounter memory allocation errors. This problem is more likely to occur on machines with a greater number of cores. Our testing has shown this to be an issue using one of our test data sets on a 32-bit machine with 12 cores. If the user encounters memory allocation errors with *bulk_extractor* they will likely see an error similar to the following:

```
bulk_extractor scan error: 'std::exception Scanner: gzip Exception:
std::bad_alloc sbuf.pos0: (121894266880) bufsize=20971520'
```

Memory allocation errors such as the one shown above will contain the phrase “bad_alloc” somewhere in the message. If the user encounters this error, they should try running *bulk_extractor* with fewer threads. For example, the following command will run *bulk_extractor* with only 4 threads (the `-j` option changes this parameter):

```
■ bulk_extractor -j 4 -o output mydisk.raw
```

Reducing the number of threads and re-running the program should eliminate the problem.

Users may encounter errors if they are processing a large disk image and trying to write the output of *bulk_extractor* to an output file directory on a smaller drive. In that case the user might see an error similiar to the following:

```

bulk_extractor version: 1.4.0
Input file: G:\nps-2011-2tb\nps-2011-2tb.E01
Output directory: C:\Users\Mark Richer\Documents\BE Testing\OFD nps-2011-2tb 64bit
Disk Size: 2000054960128
Threads: 12
DISK FULL
DISK FULL
DISK FULL
*** carve: Cannot write(pos=7,0 len=24724184): No space left on device
DISK FULL
DISK FULL
DISK FULL
DISK FULL
DISK FULL
*** carve: Cannot write(pos=7,0 len=24724198): No space left on device
*** carve: Cannot write(pos=7,0 len=49160): No space left on device

*** carve: Cannot create C:\Users\Mark Richer\Documents\BE Testing\OFD nps-2011-2tb
64bit/kml/000/426602508288-ZIP-0.kml: No space left on device

Could not make directory C:\Users\Mark Richer\Documents\BE Testing\OFD nps-2011-2tb
64bit/kml/001: No space left on device

Phase 3. Creating Histograms
Cannot open histogram output file: C:\Users\Mark Richer\Documents\BE Testing\OFD
nps-2011-2tb 64bit/ccn_track2_histogram.txt

Elapsed time: 45111.4 sec.
Overall performance: 44.3359 MBytes/sec
Total email features found: 6716934

```

If this situation is encountered, the solution is to run *bulk_extractor* with an output directory on a machine with more available disk space so that *bulk_extractor* has room to create all the output files and directories required.

12 Related Reading

There are numerous articles and presentations available related to digital forensics, specifically *bulk_extractor*, and its practical and research applications. Some of those articles are specifically cited throughout this manual. Other useful references include but are not limited to:

- Garfinkel, S. File Cabinet Forensics, Journal of Digital Forensics, Security and Law, Vol 6(4). <http://www.jdfsl.org/subscriptions/abstracts/JDFSL-V6N4-column-Garfinkel.pdf>
- Garfinkel, S. Every Last Byte. J. of Digital Forensics, Security and Law, 6:7-8. Column. <http://www.jdfsl.org/subscriptions/abstracts/column-v6n2-Garfinkel.htm>
- Phillips, Kenneth N; Aaron Pickett; Simson Garfinkel, Embedded with Facebook: DoD Faces Risks from Social Media, CrossTalk, May/June 2011. <http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA542587>
- Rowe, Neil, Schwamm, Riqui, Garfinkel, Simson. Language Translation for File Paths, DFRWS 2013, Aug 4-7, 2013. Monterey, CA. <http://www.dfrws.org/>

2013/proceedings/DFRWS2013-5.pdf

- Garfinkel, S., Nelson, A., Young, J., “A General Strategy for Differential Forensic Analysis”, DFRWS 2012, Aug. 6-8, 2012, Washington, DC. <http://www.dfrws.org/2012/proceedings/DFRWS2012-6.pdf>
- Garfinkel, S., “Lessons Learned Writing Computer Forensics Tools and Managing a Large Digital Evidence Corpus”, DFRWS 2012, Aug. 6-8, 2012, Washington, DC. <http://simson.net/clips/academic/2012.DFRWS.DIIN382.pdf>
- N. C. Rowe and S. L. Garfinkel, Finding anomalous and suspicious files from directory metadata on a large corpus. 3rd International ICST Conference on Digital Forensics and Cyber Crime, Dublin, Ireland, October 2011. In P. Gladyshev and M. K. Rogers (eds.), Lecture Notes in Computer Science LNICST 88, Springer-Verlag, 2012, pp. 115-130. <http://simson.net/clips/academic/2012.IICDFCC.Anomalous.pdf>
- Presentation - Using *bulk_extractor* for digital forensics triage and cross-drive analysis, DFRWS 2012. http://digitalcorpora.org/downloads/bulk_extractor/doc/2012-08-08-bulk_extractor-tutorial.pdf
- Presentation - Digital Signatures: Current Barriers, Invited Talk, 10th Symposium on Identity and Trust on the Internet, Gaithersburg, MD, 2011. <http://middleware.internet2.edu/idtrust/2011/slides/07-digital-signatures-current-barriers-garfinkel.pdf>
- Courrejou, Timothy and Simson Garfinkel. A comparative analysis of file carving software. Technical Report NPS-CS-11-006, Naval Postgraduate School, September 2011. <http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA550119>

Appendices

A Output of *bulk_extractor* Help Command

```
C:\>bulk_extractor -h
```

```
bulk_extractor version 1.5.0
```

```
Usage: bulk_extractor [options] imagefile
```

```
runs bulk extractor and outputs to stdout a summary of what was found where
```

Required parameters:

```
imagefile      - the file to extract
or -R filedir  - recurse through a directory of files
                  HAS SUPPORT FOR E01 FILES
-o outdir      - specifies output directory. Must not exist.
                  bulk_extractor creates this directory.
```

Options:

```
-i              - INFO mode. Do a quick random sample and print a report.
-b banner.txt  - Add banner.txt contents to the top of every output file.
-r alert_list.txt - a file containing the alert list of features to alert
                  (can be a feature file or a list of globs)
                  (can be repeated.)
-w stop_list.txt - a file containing the stop list of features (white list)
                  (can be a feature file or a list of globs)s
                  (can be repeated.)
-F <rfile>     - Read a list of regular expressions from <rfile> to find
-f <regex>     - find occurrences of <regex>; may be repeated.
                  results go into find.txt
-q nn          - Quiet Rate; only print every nn status reports. Default 0; -1 for no status at all
-s frac[:passes] - Set random sampling parameters
```

Tuning parameters:

```
-C NN          - specifies the size of the context window (default 16)
-S fr:<name>:window=NN specifies context window for recorder to NN
-S fr:<name>:window_before=NN specifies context window before to NN for recorder
-S fr:<name>:window_after=NN specifies context window after to NN for recorder
-G NN          - specify the page size (default 16777216)
-g NN          - specify margin (default 4194304)
-j NN          - Number of analysis threads to run (default 4)
-M nn          - sets max recursion depth (default 7)
-m <max>       - maximum number of minutes to wait after all data read
                  default is 60
```

Path Processing Mode:

```
-p <path>/f    - print the value of <path> with a given format.
                  formats: r = raw; h = hex.
                  Specify -p - for interactive mode.
                  Specify -p -http for HTTP mode.
```

Parallelizing:

```
-Y <o1>        - Start processing at o1 (o1 may be 1, 1K, 1M or 1G)
-Y <o1>-<o2>    - Process o1-o2
-A <off>       - Add <off> to all reported feature offsets
```

Debugging:

```
-h            - print this message
-H            - print detailed info on the scanners
-V            - print version number
-z nn        - start on page nn
-dN          - debug mode (see source code)
```

-Z - zap (erase) output directory

Control of Scanners:

-P <dir> - Specifies a plugin directory
 Default dirs include /usr/local/lib/bulk_extractor /usr/lib/bulk_extractor and
 BE_PATH environment variable

-e <scanner> enables <scanner> -- -e all enables all
-x <scanner> disable <scanner> -- -x all disables all
-E <scanner> - turn off all scanners except <scanner>
 (Same as -x all -e <scanner>)

 note: -e, -x and -E commands are executed in order
 e.g.: '-E gzip -e facebook' runs only gzip and facebook

-S name=value - sets a bulk extractor option name to be value

Settable Options (and their defaults):

-S work_start_work_end=YES Record work start and end of each scanner in report.xml file ()
-S enable_histograms=YES Disable generation of histograms ()
-S debug_histogram_malloc_fail_frequency=0 Set >0 to make histogram maker fail with memory allocations
-S hash_alg=md5 Specifies hash algorithm to be used for all hash calculations ()
-S dup_data_alerts=NO Notify when duplicate data is not processed ()
-S write_feature_files=YES Write features to flat files ()
-S write_feature_sqlite3=NO Write feature files to report.sqlite3 ()
-S report_read_errors=YES Report read errors ()
-S ssn_mode=0 0=Normal; 1=No 'SSN' required; 2=No dashes required (accts)
-S min_phone_digits=6 Min. digits required in a phone (accts)
-S carve_net_memory=NO Carve network memory structures (net)
-S word_min=6 Minimum word size (wordlist)
-S word_max=14 Maximum word size (wordlist)
-S max_word_outfile_size=100000000 Maximum size of the words output file (wordlist)
-S wordlist_use_flatfiles=NO Override SQL settings and use flatfiles for wordlist (wordlist)
-S hashdb_mode=none Operational mode [none|import|scan]
 none - The scanner is active but performs no action.
 import - Import block hashes.
 scan - Scan for matching block hashes. (hashdb)
-S hashdb_block_size=4096 Hash block size, in bytes, used to generate hashes (hashdb)
-S hashdb_ignore_empty_blocks=YES Selects to ignore empty blocks. (hashdb)
-S hashdb_scan_path_or_socket=your_hashdb_directory File path to a hash database or
 socket to a hashdb server to scan against. Valid only in scan mode. (hashdb)
-S hashdb_scan_sector_size=512 Selects the scan sector size. Scans along
 sector boundaries. Valid only in scan mode. (hashdb)
-S hashdb_import_sector_size=4096 Selects the import sector size. Imports along
 sector boundaries. Valid only in import mode. (hashdb)
-S hashdb_import_repository_name=default_repository Sets the repository name to
 attribute the import to. Valid only in import mode. (hashdb)
-S hashdb_import_max_duplicates=0 The maximum number of duplicates to import
 for a given hash value, or 0 for no limit. Valid only in import mode. (hashdb)
-S exif_debug=0 debug exif decoder (exif)
-S jpeg_carve_mode=1 0=carve none; 1=carve encoded; 2=carve all (exif)
-S min_jpeg_size=1000 Smallest JPEG stream that will be carved (exif)
-S zip_min_uncompr_size=6 Minimum size of a ZIP uncompressed object (zip)
-S zip_max_uncompr_size=268435456 Maximum size of a ZIP uncompressed object (zip)
-S zip_name_len_max=1024 Maximum name of a ZIP component filename (zip)
-S unzip_carve_mode=1 0=carve none; 1=carve encoded; 2=carve all (zip)
-S rar_find_components=YES Search for RAR components (rar)
-S raw_find_volumes=YES Search for RAR volumes (rar)
-S unrar_carve_mode=1 0=carve none; 1=carve encoded; 2=carve all (rar)
-S gzip_max_uncompr_size=268435456 maximum size for decompressing GZIP objects (gzip)
-S pdf_dump=NO Dump the contents of PDF buffers (pdf)
-S opt_weird_file_size=157286400 Weird file size (windirs)
-S opt_weird_file_size2=536870912 Weird file size2 (windirs)

-S opt_max_cluster=67108864 Ignore clusters larger than this (windirs)
-S opt_max_cluster2=268435456 Ignore clusters larger than this (windirs)
-S opt_max_bits_in_attr=3 Ignore FAT32 entries with more attributes set than this (windirs)
-S opt_max_weird_count=2 Ignore FAT32 entries with more things weird than this (windirs)
-S opt_last_year=2019 Ignore FAT32 entries with a later year than this (windirs)
-S xor_mask=255 XOR mask string, in decimal (xor)
-S sqlite_carve_mode=2 0=carve none; 1=carve encoded; 2=carve all (sqlite)

These scanners disabled by default; enable with -e:

-e base16 - enable scanner base16
-e facebook - enable scanner facebook
-e hashdb - enable scanner hashdb
-e outlook - enable scanner outlook
-e scean - enable scanner scean
-e wordlist - enable scanner wordlist
-e xor - enable scanner xor

These scanners enabled by default; disable with -x:

-x accts - disable scanner accts
-x aes - disable scanner aes
-x base64 - disable scanner base64
-x elf - disable scanner elf
-x email - disable scanner email
-x exif - disable scanner exif
-x find - disable scanner find
-x gps - disable scanner gps
-x gzip - disable scanner gzip
-x hiber - disable scanner hiber
-x httplogs - disable scanner httplogs
-x json - disable scanner json
-x kml - disable scanner kml
-x net - disable scanner net
-x pdf - disable scanner pdf
-x rar - disable scanner rar
-x sqlite - disable scanner sqlite
-x vcard - disable scanner vcard
-x windirs - disable scanner windirs
-x winlnk - disable scanner winlnk
-x winpe - disable scanner winpe
-x winprefetch - disable scanner winprefetch
-x zip - disable scanner zip

Table 1: Input Data Processed by the Scanners

Scanner Name	Data Type	Section Discussed in Manual
<i>base16</i>	Base 16 (hex) encoded data (includes MD5 codes embedded in the data)	Subsection 5.2
<i>base64</i>	Base 64 code	Subsection 4.6 and Subsection 5.2
<i>elf</i>	Executable and Linkable Format (ELF)	Subsection 5.1
<i>exif</i>	EXIF structures from JPEGs (and carving of JPEG files)	Subsection 5.5
<i>gzip</i>	GZIP files and ZLIB-compressed GZIP streams	Subsection 4.6 and Subsection 5.2
<i>aes</i>	In-memory AES keys from their key schedules	Subsection 5.2
<i>json</i>	JavaScript Object Notation files and objects downloaded from web servers, as well as JSON-like objects found in source code	Subsection 5.1
<i>jpeg</i>	JPEG carving. Default is only encoded JPEGs are carved. JPEGs without EXIFs are also carved	Subsection 4.3 and Subsection 5.5
<i>kml</i>	KML files (carved)	Subsection 5.3
<i>rar</i>	RAR components in unencrypted archives are decrypted and processed. Encrypted RAR file are carved.	Subsection 4.3
<i>pdf</i>	Text from PDF files (extracted for processing not carved)	Subsection 4.6
<i>windirs</i>	Windows FAT32 and NTFS directory entries	Subsection 5.2
<i>hiber</i>	Windows Hibernation File Fragments (decompressed and processed, not carved)	Subsection 4.6
<i>winprefetch</i>	Windows Prefetch files, file fragments (processed)	Subsection 5.1
<i>winpe</i>	Windows Preinstallation Environment (PE) Executables (.exe and .dll files notated with MD5 hash of first 4k)	Subsection 5.1
<i>vcard</i>	vCard files (carved)	Subsection 5.3
<i>gps</i>	XML from Garmin GPS devices (processed)	Subsection 5.3
<i>zip</i>	ZIP files and zlib streams (processed, and optionally carved)	Subsection 4.3 and Subsection 4.6

Table 2: There are three carving modes in *bulk_extractor* that are specified separately for each file type, JPEG, ZIP and RAR.

Mode	Mode Description
0	Do not carve files of the specified type.
1	Only carve encoded files of the specified type
2	Carve everything of the specified type.

Table 3: The kinds of encodings that can be decoded by *bulk_extractor* and the amount of context required for the decoding

Encoding	Can be decoded when <i>bulk_extractor</i> finds
GZIP	The beginning of a zlib-compressed stream
BASE64	The beginning of a BASE64-encoded stream
HIBER	Any fragment of a hibernation file can generally be decompressed, as each Windows 4k page is separately compressed and the beginning of each compressed page in the hibernation file is indicated by a well-known sequence
PDF	Any PDF stream compressed with ZLIB bracketed by <i>stream</i> and <i>endstream</i>
ZIP	The local file header of a ZIP-file component