Chapter 11
Keyword Forensics

Learning Objectives
The objectives of this chapter are to:

- Understand the forensic technique of Keyword Searching
- Understand concepts of the Regular Expression
- Become familiar with the tools involved in keyword searching process, including srch_strings, grep, blkcat, ifind, and istat

Keyword searching is a common technique used in forensic investigation to quickly examine a disk image or data archive acquired from a suspect's computer and narrow down the region of interest within files, deleted data, and slack spaces. This is accomplished by traversing through a hard disk image using a known keyword (e.g. pornography, confidential); thus, locating the allocated spaces dedicated to the suspicious file and then retrieving it. It is typically completed at the early state of digital investigation in order to create a foundation of where the investigation should start.

For example law enforcement recovered the laptop computer used by Boston bombing suspect Dzhokhar Tsarnaev [1]. They had to scour through the contents of the laptop to determine if it contains any evidence pointing Dzhokhar and his accomplice, Tamerlan Tsarnaev (Dzhokhar's brother) to Boston bombing and there is any indication of terrorism. Using keyword searches could be very effective instead of manually going through all the files on the laptop.

Actually, searching keywords of interest is very common for computer users too. This can be done using search function provided in OSs, such as Windows Search, and applications, such as Microsoft Outlook. The most intuitive method for keyword searching is to provide a single keyword, and have the tool search for occurrences of that keyword within a document. This is basically how the Linux grep tool works. While it seems fairly straightforward to perform keyword search—enter keywords of

© Springer Nature Switzerland AG 2018

interest you want to search for and then perform the search, a forensic keyword search process is much more complicated than a regular keyword search in our daily computer use. For example, the text or files can be encoded in Unicode so they have to be decoded to extract printable characters (or strings). Also, we may have to search a pattern of strings instead of an exact match due to many reasons. On the one hand, the suspect may misspell or mistype a word. On the other hand, several meaningful sentences can express the same thing and feelings. Thus, we need to search using a Regular Expression in order to retrieve any string that approximately matches a given search pattern.

In this chapter, we'll learn forensic keyword search process, how to use the tools that are openly available for keyword searching. Also, we will learn how to create a repository of "dirty word" keyword(s) to search the disk image from, how to find the questionable file's meta-data structure location that has allocated the given disk unit (or data unit) based on the search results, and how to obtain detail information on the meta-data structure of the file.

11.1 Forensic Keyword Searching Process

The forensic keyword search process typically follows the workflow shown in Fig. 11.1, where an analyst acquire a questionable/volatile hard drive and creates a repository of keyword(s), also known as "dirty words", to search the disk image [6]. However, it's a challenge to find a string type keyword when the hard image disk is

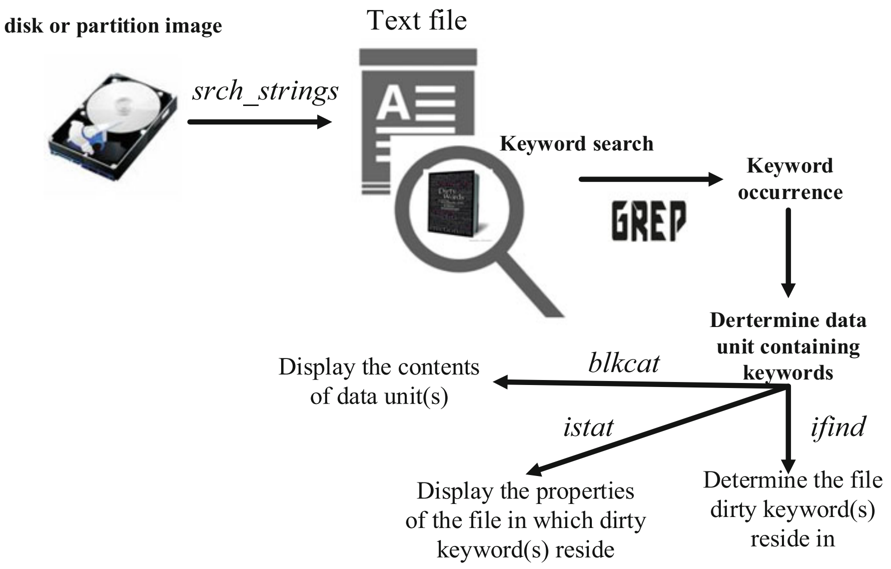

disk or partition image
Text file
srch_strings
Keyword search
GREP
Keyword occurrence
Dertermine data unit containing keywords
blkcat
Display the contents of data unit(s)
istat
Display the properties of the file in which dirty keyword(s) reside
ifind
Determine the file dirty keyword(s) reside in

Fig. 11.1 Keyword search process overview

comprise of binary data. Therefore, we first need to extract printable data from a binary image disk, for example, using TSK's srch_strings command [5], where the .asc file outputted (a text file) is then used to find all the occurrences of keywords. The output .asc file contains all the printable data along with their locations in the disk image. Afterwards, we can search the keyword within the .asc file, for example, using the grep command. If a match is found, the analyst perform further analysis by discovering the meta-data structure for the file that occupies the disk unit where keyword resides. The next section goes into extensive detail on how grep functions. Other noteworthy functions involved includes:

- "blkcat" used to display contents of data unit containing keywords;
- "ifind" used to find meta-data structure that allocates or points to a given data unit;
- "istat" used to display details of a given meta-data structure.

Henceforth, analyst can view data by either (a) retrieving the data unit that contains the dirty keywords (using blkcat) or (b) figuring out which file dirty keyword(s) reside in (using ifind) and the details of the file meta-data structure (using istat).

11.2 Grep and Regular Expressions

The grep (stand for "Globally search a Regular Expression and Print") command is a Linux tool used to find input files (or standard input when there is no file to name) for a given line of data. It matches based on a regular expression, which is a method for specifying a set of strings. The basic usage of grep command is to search for a specific string, represented by a regular expression in specified file(s), and following are examples of **grep** commands that can be executed [2] (Table 11.1):

Regular expression provides a basic and extended standard syntax for creating patterns designed specifically to lookup a set of strings from a list of elements or to verify if a given string follows a particular arrangement (e.g. postcode, email address, phone number, ect) [7]. Literally, Basic Regular Syntax (BRE) and Extended Regular Syntax (ERE) work together. However, BRE requires that the metacharacters ( ) and { } be designated and \{\}, whereas ERE does not [3]. Also, ERE introduces more metacharacters, including "?", "+", and "|".

For example, a basic regular expression "[a-z]" matches any single lowercase character while an extensive regular expression "/^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$/" matches "http://", "https://", or neither of them, followed by a series of digits and letters, followed by a single dot and more digits and letters after another single dot, finally followed by a single "/". A break down of the later example is shown in Fig. 11.2, which can be observed that it matches urls. Please refer the appendix located at the end of this chapter to view a table describing each meta-characters.

Table 11.1 Examples of how to use grep [2]

grep forensics files	{search *files* for lines with “forensics”}
grep 'forensics?' files	{search *files* for lines with “forensics” or “forensic”}
grep '^forensics' files	{“forensics” at the start of a line}
grep 'forensics$' files	{“forensics” at the end of a line}
grep '^forensics$' files	{lines containing only “forensics”}
grep '[Ff]orensics' files	{search for “Forensics” or “forensics”}
grep '\^f' files	{search *files* for lines with “^f”, “\” escapes the ^}
grep '^$' files	{search for blank lines}
grep '[0-9][0-9][0-9]' files	{search for triples of numeric digits}
grep—f dws.txt files	{The—f option specifies a file where grep reads patterns. In this example, the search patterns are contained in a file called dws.txt, one per line}

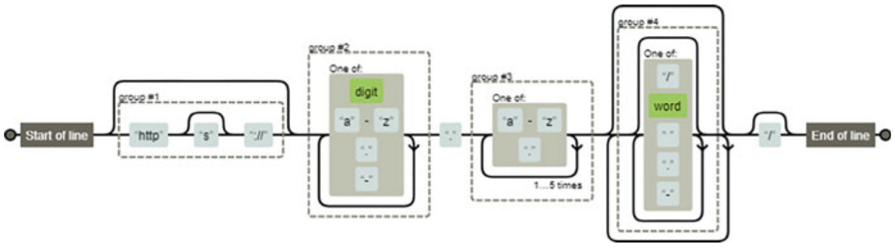
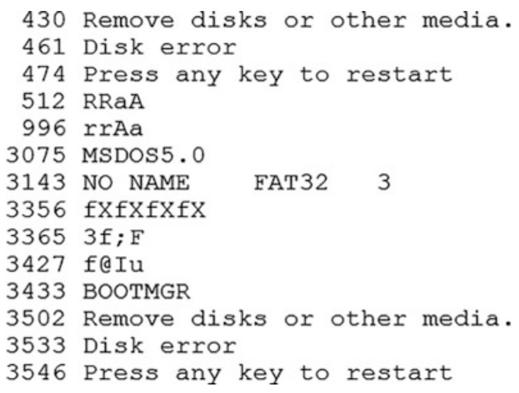


Fig. 11.2 Regular express brake down of “/^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$/” [4]

11.3 Case Study

In the following case, we assume that authorities confiscate a suspect’s disk and you are asked to analyze it using its disk image provided by the law authority. Suppose that the image called “thumbimage_fat.dd” provided with this book is the disk image provided to you. Your mission is to find out whether or not it contains sensitive data and information. If so, you will have to go further and discover more details related to the keyword, for example, which data unit containing keywords, which file containing keywords if applicable. For ease of illustration, we assume that a keyword “**overview**” is the sensitive data which we are interested in.

As shown in Fig. 11.1, we first need to extract printable data from a binary image disk using TSK’s srch_strings command. For simplicity, we only analyze the partition in the disk image. Thus, we will extract the partition from “thumbimage_fat.dd” using *mmls* and *dcfldd*. You may refer to Chap. 4 on how to



```
 430 Remove disks or other media.
 461 Disk error
 474 Press any key to restart
 512 RRaA
 996 rrAa
3075 MSDOS5.0
3143 NO NAME    FAT32   3
3356 fXfXfXfX
3365 3f;F
3427 f@Iu
3433 BOOTMGR
3502 Remove disks or other media.
3533 Disk error
3546 Press any key to restart
```

Fig. 11.3 Example of part of output ascii file

use mmls tool to discover the layout of the disk and then use dcfldd to extract a partition. Suppose that the image “fatimage.dat” is the partition extracted. Next, we extract printable data from “fatimage.dat” using the following command

```
srch_strings -t d fatimage.dd > fat-kw.ascii.str
```

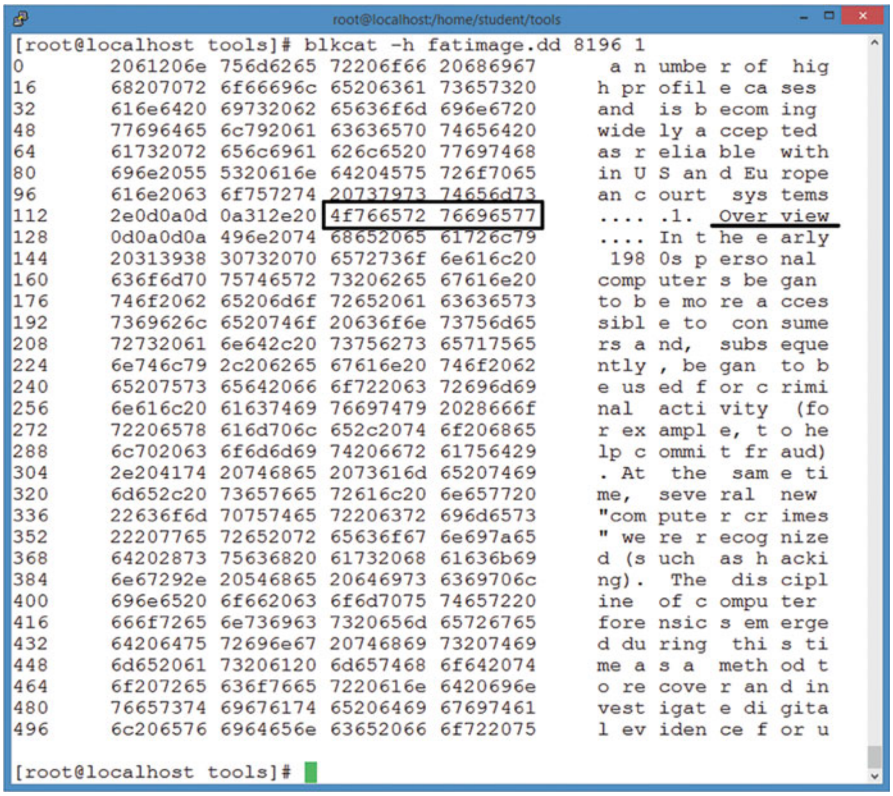
where the “-t d” option specifies a location for the discovered string to be output and the location is using byte offset in decimal from the beginning of the partition (or the FAT file system in this example). Figure 11.3 show part of the output ascii file, each line containing a byte offset (decimal) and its corresponding string found there.

Now we can use grep to search keywords we are interested in. In our example, we will search a particularly word “overview” using the following command. Note that search should be case insensitive here.

```
grep -i overview fat-kw.ascii.str
   4196469 1. Overview
```

where the “-i” specifies that the matching will be case insensitive. It can be observed that the word “Overview” appears in a string located at the byte offset 4196469. Nevertheless, hard disk uses sector address to locate an area on disk, whereas a file system uses cluster or block number to identify a data unit on disk. Thus, we need to convert byte offset to sector address and then cluster or block address. Regarding conversion of byte offset to sector address in a partition, you can divide the offset by the sector size i.e. 512 bytes and determine the sector address by obtaining the floor (rounded down) integer number of the quotient. Thus, we have

$$\text{sector address} = \text{floor}(4196469/512) = 8196$$



```
[root@localhost tools]# blkcat -h fatimage.dd 8196 1
0       2061206e 756d6265 72206f66 20686967      a n umbe r of  hig
16      68207072 6f66696c 65206361 73657320     h pr ofil e ca ses
32      616e6420 69732062 65636f6d 696e6720     and  is b ecom ing
48      77696465 6c792061 63636570 74656420     wide ly a ccep ted
64      61732072 656c6961 626c6520 77697468     as r elia ble  with
80      696e2055 5320616e 64204575 726f7065     in U S an d Eu rope
96      616e2063 6f757274 20737973 74656d73     an c ourt  sys tems
112     2e0d0a0d 0a312e20 4f766572 76696577     .... .1.  Over view
128     0d0a0d0a 496e2074 68652065 61726c79     .... In t he e arly
144     20313938 30732070 6572736f 6e616c20      198 0s p erso nal
160     636f6d70 75746572 73206265 67616e20     comp uter s be gan
176     746f2062 65206d6f 72652061 63636573     to b e mo re a cces
192     7369626c 6520746f 20636f6e 73756d65     sibl e to  con sume
208     72732061 6e642c20 73756273 65717565     rs a nd,  subs eque
224     6e746c79 2c206265 67616e20 746f2062     ntly , be gan  to b
240     65207573 65642066 6f722063 72696d69     e us ed f or c rimi
256     6e616c20 61637469 76697479 2028666f     nal  acti vity  (fo
272     72206578 616d706c 652c2074 6f206865     r ex ampl e, t o he
288     6c702063 6f6d6d69 74206672 61756429     lp c ommi t fr aud)
304     2e204174 20746865 2073616d 65207469     . At  the  sam e ti
320     6d652c20 73657665 72616c20 6e657720     me,  seve ral  new
336     22636f6d 70757465 72206372 696d6573     "com pute r cr imes
352     22207765 72652072 65636f67 6e697a65     " we re r ecog nize
368     64202873 75636820 61732068 61636b69     d (s uch  as h acki
384     6e67292e 20546865 20646973 6369706c     ng).  The  dis cipl
400     696e6520 6f662063 6f6d7075 74657220     ine  of c ompu ter
416     666f7265 6e736963 7320656d 65726765     fore nsic s em erge
432     64206475 72696e67 20746869 73207469     d du ring  thi s ti
448     6d652061 73206120 6d657468 6f642074     me a s a  meth od t
464     6f207265 636f7665 7220616e 6420696e     o re cove r an d in
480     76657374 69676174 65206469 67697461     vest igat e di gita
496     6c206576 6964656e 63652066 6f722075     l ev iden ce f or u

[root@localhost tools]#
```

Fig. 11.4 Example of output of *blkcat* tool

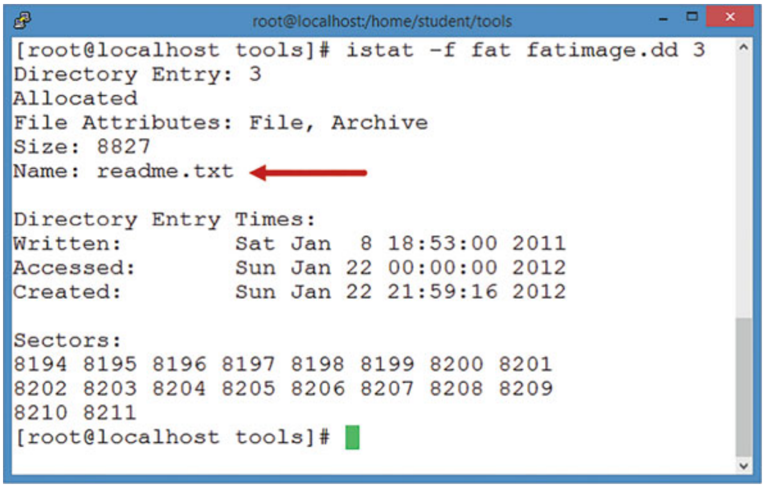
where floor() is floor function, which outputs the largest integer less than or equal to the input.

Now we know the word "overview" resides in a sector whose address is 8196. Henceforth, we will conduct a more in-depth investigation. First, we can view the contents of data unit (or a sector here) using blkcat command. It can be evident in Fig. 11.4 that the word "Overview" is found at byte offset 120–127.

Next, let's figure out which file the word resides in. First, we can find the metadata structure that has allocated the above disk unit using the following command.

```
ifind -f fat -d 8196 fatimage.dd
3
```

Second, we can find the name of the file (or directory) using the above metadata structure 3 using the following command.



```
[root@localhost tools]# istat -f fat fatimage.dd 3
Directory Entry: 3
Allocated
File Attributes: File, Archive
Size: 8827
Name: readme.txt

Directory Entry Times:
Written:        Sat Jan  8 18:53:00 2011
Accessed:       Sun Jan 22 00:00:00 2012
Created:        Sun Jan 22 21:59:16 2012

Sectors:
8194 8195 8196 8197 8198 8199 8200 8201
8202 8203 8204 8205 8206 8207 8208 8209
8210 8211
[root@localhost tools]#
```

Fig. 11.5 Example of output of *istat* tool

```
ffind fatimage.dd 3
/readme.txt
```

It can be observed that a file called “readme.txt” in the root directory contains the word “Overview”.

Or, we can display the details of the file meta-data structure using *istat* command (Fig. 11.5).

Review Questions

1. Assuming a text file contains five lines. Each line starts with “line”, followed by its line number. For example, the first line starts with “line1”. What is the number of lines in the file which match with the pattern line[1–3]?

   (a) 0
   (b) 1
   (c) 2
   (d) 3

2. Write a command that uses grep with regular expressions to search a particular word “forensics” using case insensitivity in all files in the current directory?
3. Write a command that uses grep with regular expressions to look for computer science course number, which starts with a string “cs” followed by a triple of numeric digits, at the beginning of a line in a file called ‘csprogram.txt’.
4. Which of the following regular expressions will match the pattern “forensics” only at the beginning of a line?

(a) ^forensics
(b) forensics^
(c) forensics
(d) All of the above

5. Which of the following TSK tools is used to print the strings of printable characters in files?

(a) blkcat
(b) ifind
(c) istat
(d) srch_strings

11.4 Practice Exercise

The objective of this exercise is to learn how to do forensic keyword research with TSK.

11.4.1 Setting Up Practical Exercise Environment

For this exercise, we will use a disk image ("thumbimage_fat.dd") provided with this book and will need to upload this disk image to Forensic Workstation we have built up in Chap. 3. Also, we need to extract a partition (or the only partition) from the disk image, and the partition is formatted with an FAT file system.

11.4.2 Exercises

Part A: Extract the Partition(s) from Disk Image
Extract all the partitions from the USB drive image "**thumbimage_fat.dd**" provided in the book by using the *dcfldd* tool.

Hint: There is only one partition in the above USB drive image, and you have to know the starting point and length of a partition in order to extract it. You can use TSK's mmls tool to determine the layout of a disk, particularly the locations of its partitions.

Q1. Writing down your command(s) issued to extract the partition from the USB drive image "thumbimage_fat.dd"?

Q2. What is the file system type for the extracted partition?

Q3. What is the size of the file system for the extracted partition? (in **MB**)

Part B: File System Layer Analysis

Retrieve the details associated with the file system on the extracted partition above.

Hint: To retrieve the details associated with a file system, you can use the "fsstat" tool in TSK.

Q4. What is the cluster size? (in **KB**)

Q5. What is the Volume ID?

Q6. What is the amount of disk space for the FAT file system? (in **KB**)

Part C: Searching for Keywords

In order for a disk or partition image to be searched by using a search tool like "**grep**", we will need to print the strings of printable characters in the disk image into a text file, and then a search can be performed against the text file instead of the image file. TSK provides a tool called "**srch_strings**" to print the strings of printable characters in files, and investigator will also need to print the location of the string so that the location (or the **byte offset**) can be used later to locate the data unit which contains any keywords of interest to the investigation.

Afterwards, you can search the resulted text file, based on your defined keywords by using the "**grep**" command. It is worth noting that if we were to simply perform a **grep** on the image we would not have made any of these hits at all. Thus, we will search the text file resulted from the command "**srch_strings**".

For illustrative purposes, we assume that "Wikipedia" is the dirty word we are interested in. Then, you can define the keywords of interest to you as your "dirty word" file by adding the following keywords into a file called **"dirtywords.txt"**.

Wikipedia

If hits were found, the locations of these hits are also known. The locations are in byte offset, but we need to know an address of the disk unit (or sector/block/cluster number) in order to use TSK to perform any further investigation including:

(a) Displays the contents of a data unit containing the keywords, which can be achieved by using the TSK tool "**blkcat**";
(b) Find the meta-data structure that has allocated a given disk unit, which can be achieved by using the TSK tool "**ifind**";
(c) Display details of the meta-data structure of the file, which contains the data unit. It can be achieved by using the TSK tool "**istat**";

Answer the Following Questions

Q7. How many hits?

Q8. Select one hit and record its byte offset in decimal.

Q9. What is the sector address of the data unit where the keywords reside?

Q10. Convert the above sector number to a cluster number (or address).

Q11. Write down the full command issued to view the contents of the data unit where the keywords reside.

Q12. Write down the full command issued to find the meta-data structure that has allocated the given disk unit where the keywords reside.

Q13. What is the name of the file that occupies the data unit where the keywords reside?

Q14. What is the size of the file that occupies the data unit where the keywords reside? (in **bytes**)

Q15. How many clusters are occupied by the file that occupies the data unit where the keywords reside?

Q16. What is the slack space of the file that occupies the data unit where the keywords reside? (in **bytes**)

Appendix: Regular Expression Metacharacters [2, 4]

Metacharacter	Description
^	Matches the following item at the beginning of a text line
$	Matches the preceding item at the end of a text line
.	Matches any single character
[. . .]	A bracket expression. Matches a single character in the bracketed list or range
[^. . .]	Matches a single character that is not contained within the brackets
( )	Defines a marked subexpression. A marked subexpression is also called a block or capturing group. BRE mode requires
*	Matches the preceding item zero or more time
{m}	The preceding item is matched exactly m times. BRE mode requires \{*m*\}
{m,}	The preceding item is matched N or more times. BRE mode requires \{*m*,\}
{m,n}	Matches the preceding item at least *m* and not more than *n* times. BRE mode requires \{*m*,*n*\}
\	The escape of special meaning of next character

The next three metacharacters are only for extended regular expression

Metacharacter	Description
?	Matches the preceding character, metacharacter, or expression zero or one time
+	Matches the preceding character, metacharacter, or expression one or more times. There is no limit to the amount of times it can be matched
|	Matches the character, metacharacter, or expression on either side of it

Note that to use the grep command to search for metacharacters, you have to use a backslash (\) to escape the metacharacter. For example, the regular expression "^\." matches lines that start with a period.

References

1. FBI agent: Tsarnaev's computer contained extremist materials. http://www.chicagotribune.com/news/nationworld/chi-boston-bombing-suspect-computer-extremist-materials-20150319-story.html
2. http://www.robelle.com/smugbook/regexpr.html
3. https://en.wikipedia.org/wiki/Regular_expression
4. https://www.sitepoint.com/regexper-regular-expressions-explained/
5. srch_strings. http://man.he.net/man1/srch_strings
6. Keyword searching and indexing of forensic images. http://pyflag.sourceforge.net/Documentation/articles/indexing/index.html
7. http://www.zytrax.com/tech/web/regex.htm