

Digital Forensics Laboratory- Exercise 1

Disk Imaging

Objective:.....	2
Basic Concepts:	2
Disk Image Sources:.....	2
Methods for Creating Forensic Image:	3
Image File Formats:	3
Raw Images:.....	4
AFF: (Advanced Forensic Format):	4
Encase Evidence File Images (E01):.....	4
Live CD Distributions:.....	6
Experimental Procedure:	6
Step 1 – Boot Kali Linux:.....	6
Step 2 – Recognize the devices plugged onto the machine.....	8
Step 3: Explore Image Creation Tools	9
Final Task :-	18

Objective:

To explore the process of disk imaging using tools available in Kali Linux Distribution. In this experiment, we create and examine the hard disk images.

Basic Concepts:

A **disk image** is a bit by bit copy of a full disk or a single partition from a disk. Because the contents of a disk are constantly changing on a running system, disk images are often created following an intrusion or incident to preserve the state of a disk at a particular point in time.



Why bit by bit copy is needed as opposed to copying files?

What are **write blockers**? Why are they used during Forensic Imaging?

Disk Image Sources:

The physical disk usually contains **master boot record** and a partition with **logical volumes**; they are formatted with a file system. The Master boot record contains metadata about the layout of the disk including where the partition begin and end.

Image File is exact bit by bit copy of the entire source disk. The source disk could be the entire disk or one or more logical volumes

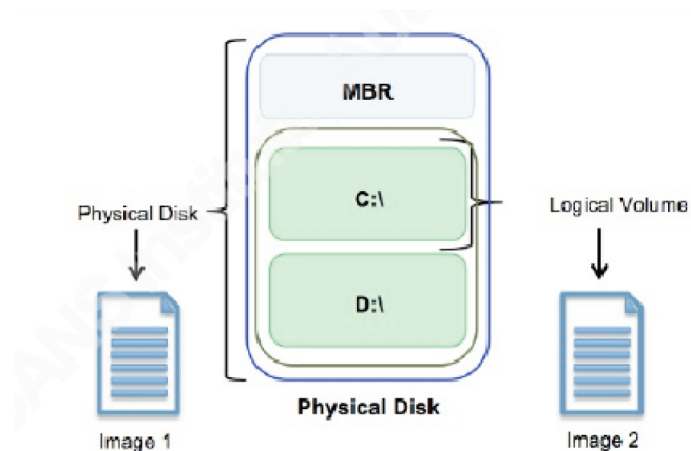


Figure 1 Layout of Hard disk

Methods for Creating Forensic Image:

There are two methods to take a forensic image. The first is to **clone** a drive; the second is to take an **image**.

The forensic image is a duplicate of a physical drive that is written to a file. The file could reside anywhere; internal disk or network storage server. The clone is a bit by bit copy of a physical disk onto **raw disk**.

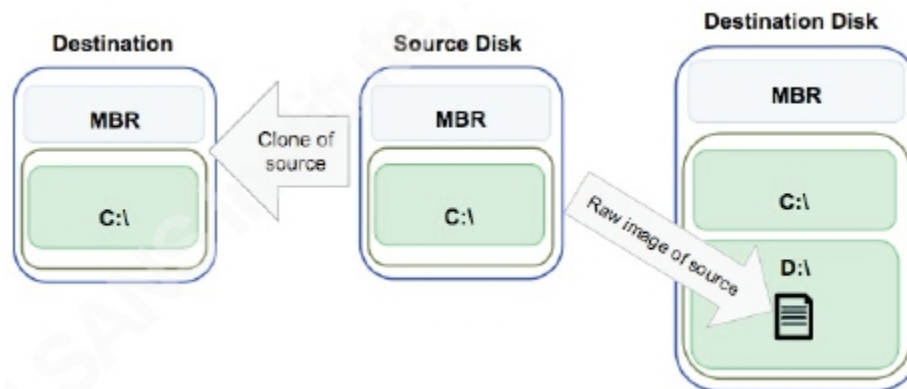


Figure 2 Process of Imaging vs Cloning

The Forensic images are usually **compressed** to save space and are separated into multiple files called **split images** that are more manageable. The images created using forensic tools usually contain metadata about the image such as timestamp when the image was created and a **cryptographic hash** acting as fingerprint of the image.



What are the smart imaging techniques employed by tools to speed up the process of imaging?

Image File Formats:

There are many file formats which are used in practice. Some are independent of any forensics package such as raw, aff, aff4 and gzip; and others are developed for use with specific forensic tools such as Encase E01, SMART, ProDiscover image file format etc. The following section briefly discusses on few widely used image formats: raw images, AFF and E01

Raw Images:

A Raw Image contains only the data from the source disk. There is no header or metadata included in the image file but some images could include a separate file with the metadata. There are several utilities which create raw images, the following extensions are the frequently used : .dd, .raw, .img.



Figure 3 Raw Image Format

AFF: (Advanced Forensic Format):

It is an extensible open format for the storage of disk images and related forensic meta-data. It is a segmented Archive File Specification where each file consists of file header followed by one or more file segments and ends with special directory segment that lists all the segments along with its offset from the file's start. These segments store all information including image itself and its metadata. There are also **metadata segments** available in an AFF file which hold information about disk image and **data segments called pages**, which holds image disk information itself.

AFFLIB is an implementation of AFF in C/C++. It comes with set of tools **aimage** (Advanced Disk Imager) which is used for creating AFF images, **afxml**; a program for converting AFF images into XML and **afconvert**; a program which convert raw images into AFF and back.

Encase Evidence File Images (E01):

The popular commercial forensics suite, EnCase; developed a proprietary format called Encase Evidence File Format. These files use file extension E01 and are based on **Expert Witness Format by ASR data**. These files are commonly referred as Expert Witness (E01) files or **EWF files**.

E01 files have both header and footer containing metadata about the image. E01 files have file integrity check calculations called Cyclic Integrity check calculation at every 64 KB intervals throughout image. The cryptographic hash called acquisition hash is calculated over data blocks only and is appended at the end of the image.

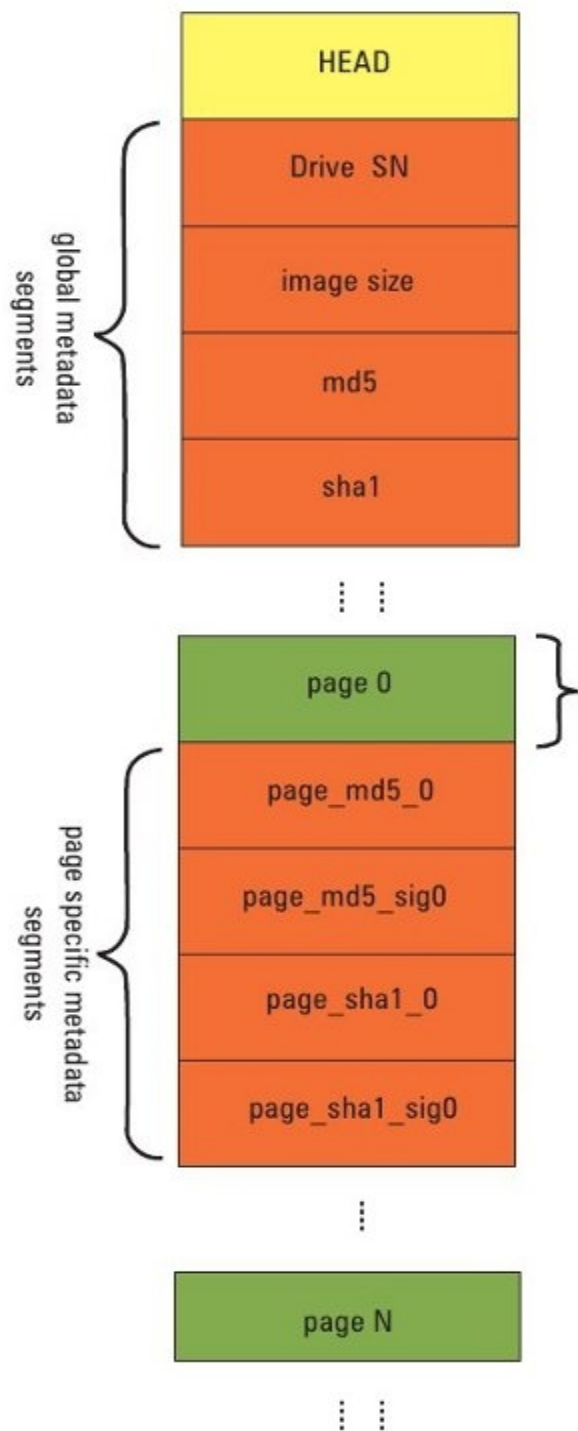


Figure 4 AFF File Organization

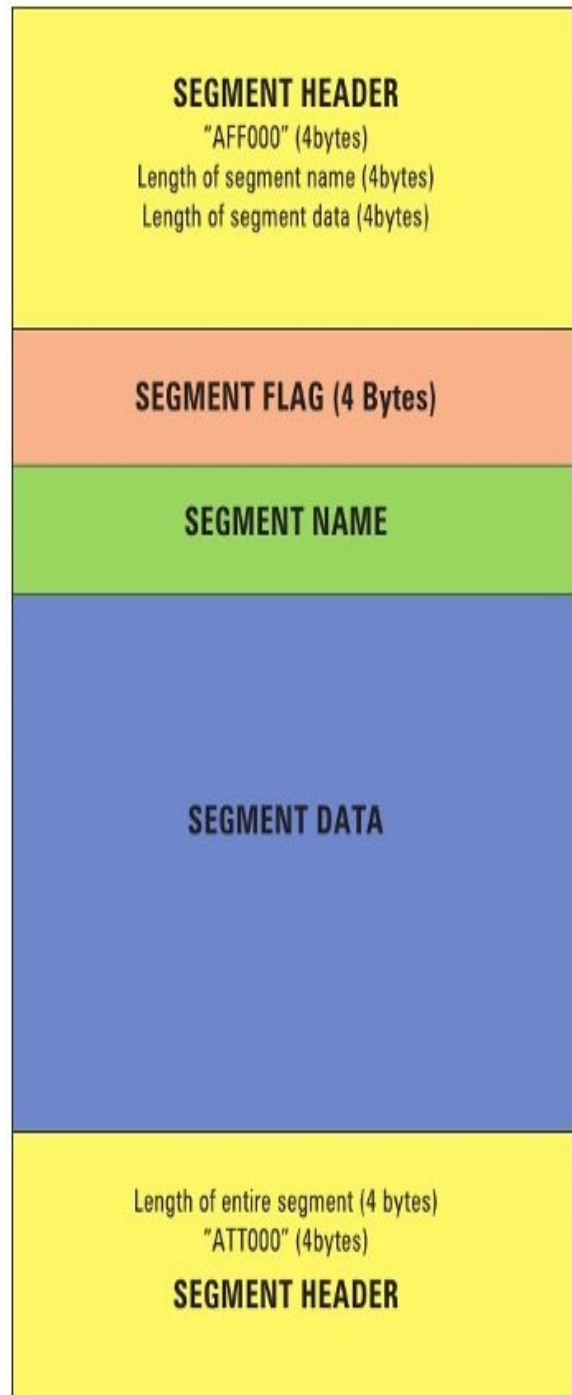


Figure 5 AFF file segment Organization

By default, the blocks are compressed using zlib compression algorithm so that after a block of data is read from the source , CRC is calculated and is appended to the block. The block is then compressed and added to the image file.

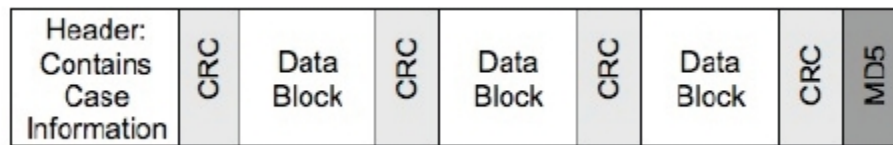


Figure 6 Encase File Format



What are the other commonly used Forensic Image File Formats?

What is the difference between the formats E01 and Ex01 ?

Tools and Techniques for Forensic Disk Imaging

In this experiment, Kali Linux; an open source Live Linux distributions is used which are specifically tailored for digital forensics.

Live CD Distributions:

A Live CD is a complete bootable computer installation including operating system which runs in the computer memory. A Live ISO image is the ISO image of the Live CD which can be used in virtual machine environments. There are many Live Linux Distributions are used in the field of forensics such as Kali, Caine, Deft, Helix and Knoppix



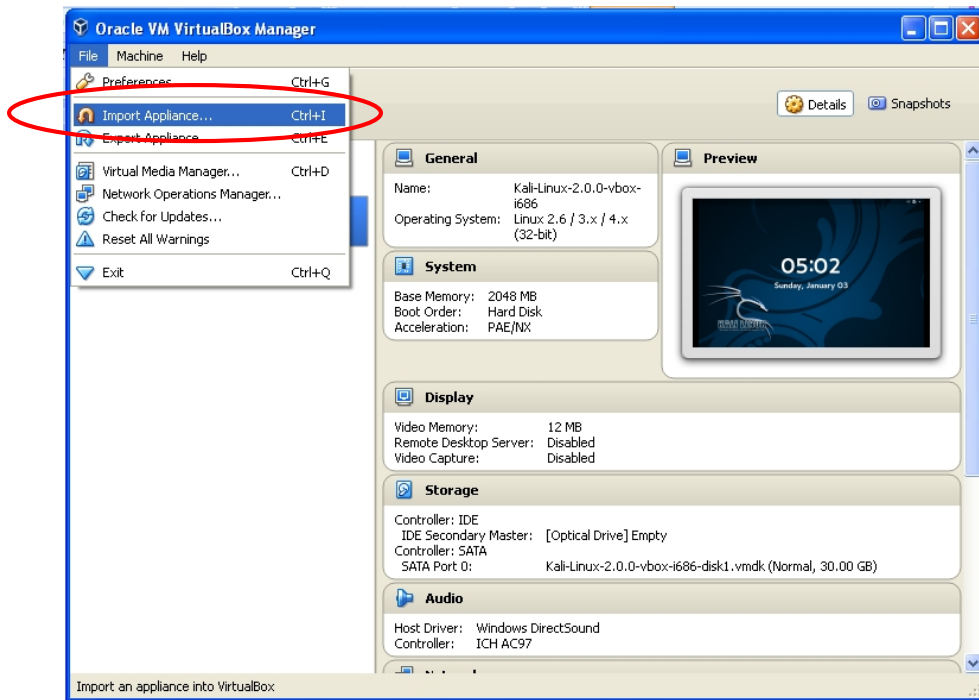
What happens while booting a Live ISO image?

What is **booting in Forensic mode**?

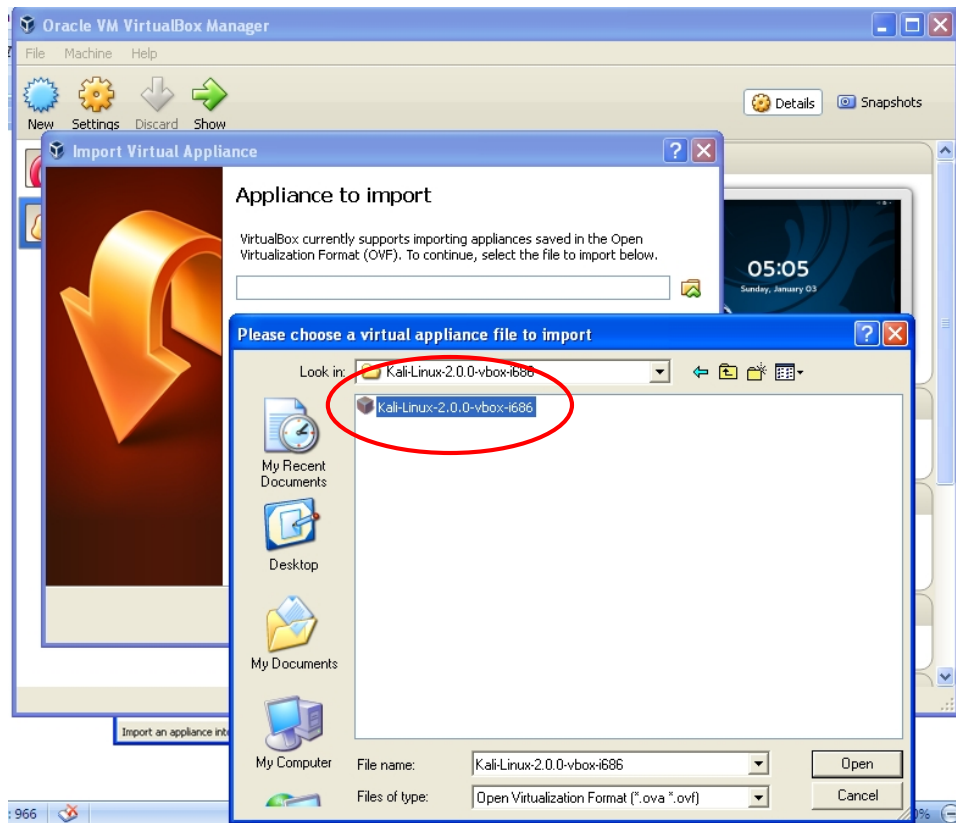
Experimental Procedure:

Step 1 – Boot Kali Linux:

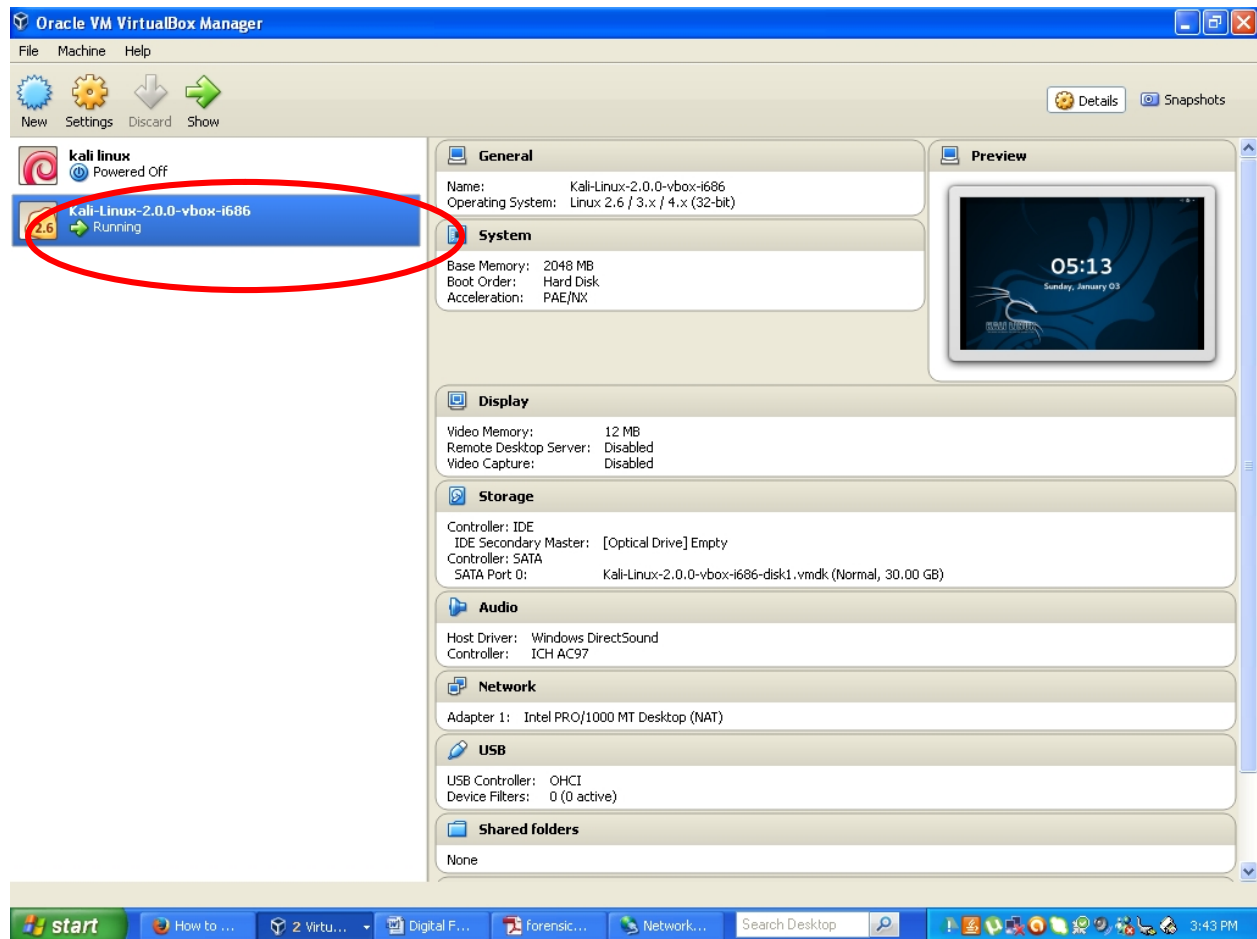
For the experiment, Kali linux distribution is booted as a virtual machine inside Virtual Box. Kali linux (32 –bit distribution) from its official website is downloaded and is available in department repository. The distribution comes with ova extension; it is a standard for packaging and distributing virtual applications. To use this, Click File -> Import Appliance



Next, choose the ova file of kali linux distribution.



The next window will show the configuration of the current virtual appliance. Scroll through the configuration list and double click on any item (or check/uncheck the box) to make changes to it. Lastly, click “Import”.



Find the Virtual Machine in the Left Pane of the Oracle VM Virtual Box Manager. Start the Virtual Machine. Default Username is “root” and Password is “toor”.

To enable USB, click Devices -> USB -> select on the USB to be recognized.

Step 2 – Recognize the devices plugged onto the machine

Since the Linux is booted inside Virtual Environment, it cannot recognize the internal hard drive. To recognize USB devices, they have to be selected as mentioned before.

Once they are selected fdisk -l command can be used for identifying various drives, their sizes and allocation tables. Open the terminal and type in the command fdisk -l


```
root@kali: ~
File Edit View Search Terminal Help
Disk identifier: 0xff9eadca

Device Boot Start End Sectors Size Id Type
/dev/sda1 * 2048 60262399 60260352 28.8G 83 Linux
/dev/sda2 60264446 62912511 2648066 1.3G 5 Extended
/dev/sda5 60264448 62912511 2648064 1.3G 82 Linux swap / Solaris

root@kali:~# fdisk -l

Disk /dev/sda: 30 GiB, 32212254720 bytes, 62914560 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xff9eadca

Device Boot Start End Sectors Size Id Type
/dev/sda1 * 2048 60262399 60260352 28.8G 83 Linux
/dev/sda2 60264446 62912511 2648066 1.3G 5 Extended
/dev/sda5 60264448 62912511 2648064 1.3G 82 Linux swap / Solaris

Disk /dev/sdb: 14.6 GiB, 15664676864 bytes, 30595072 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x00000000

Device Boot Start End Sectors Size Id Type
/dev/sdb1 32 30595071 30595040 14.6G c W95 FAT32 (LBA)

root@kali:~#
```

Space reserved
for vmdk

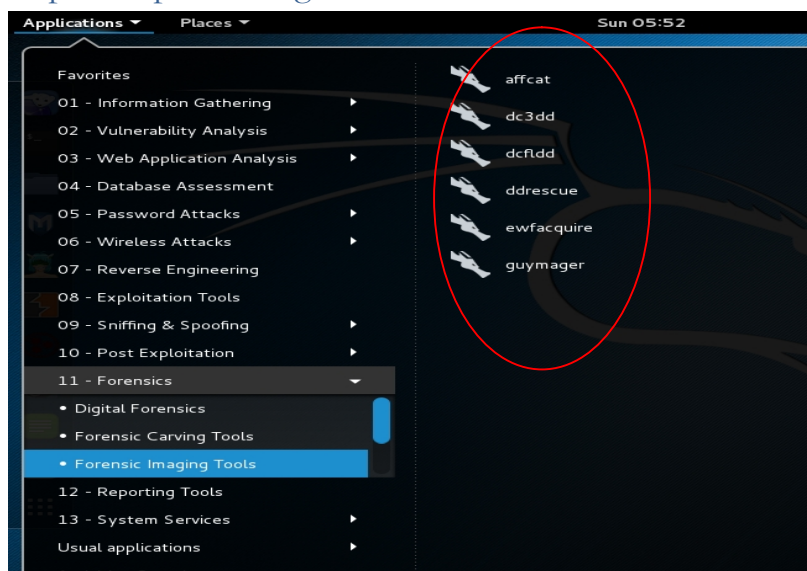
It corresponds
to external
USB drive



Describe the information presented by fdisk command ?

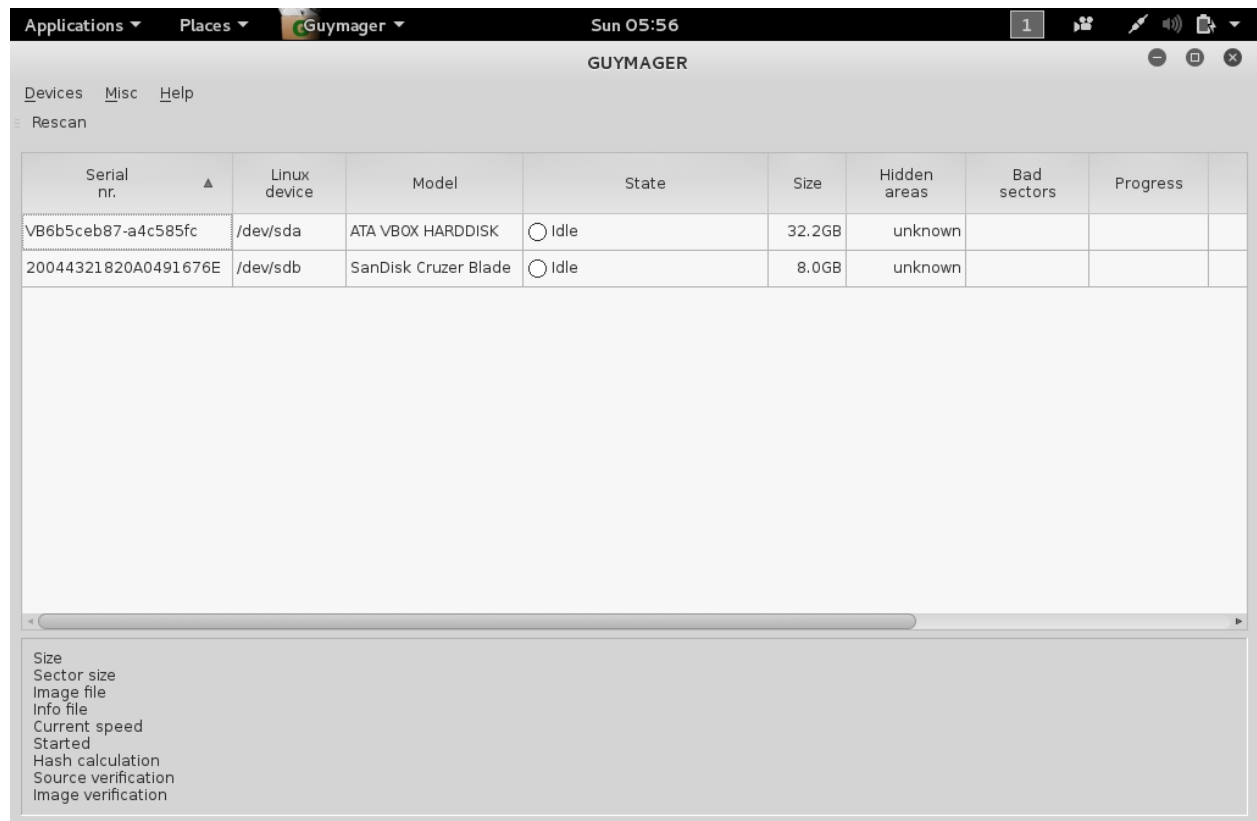
/dev/sd stands for?

Step 3: Explore Image Creation Tools



Guymager:

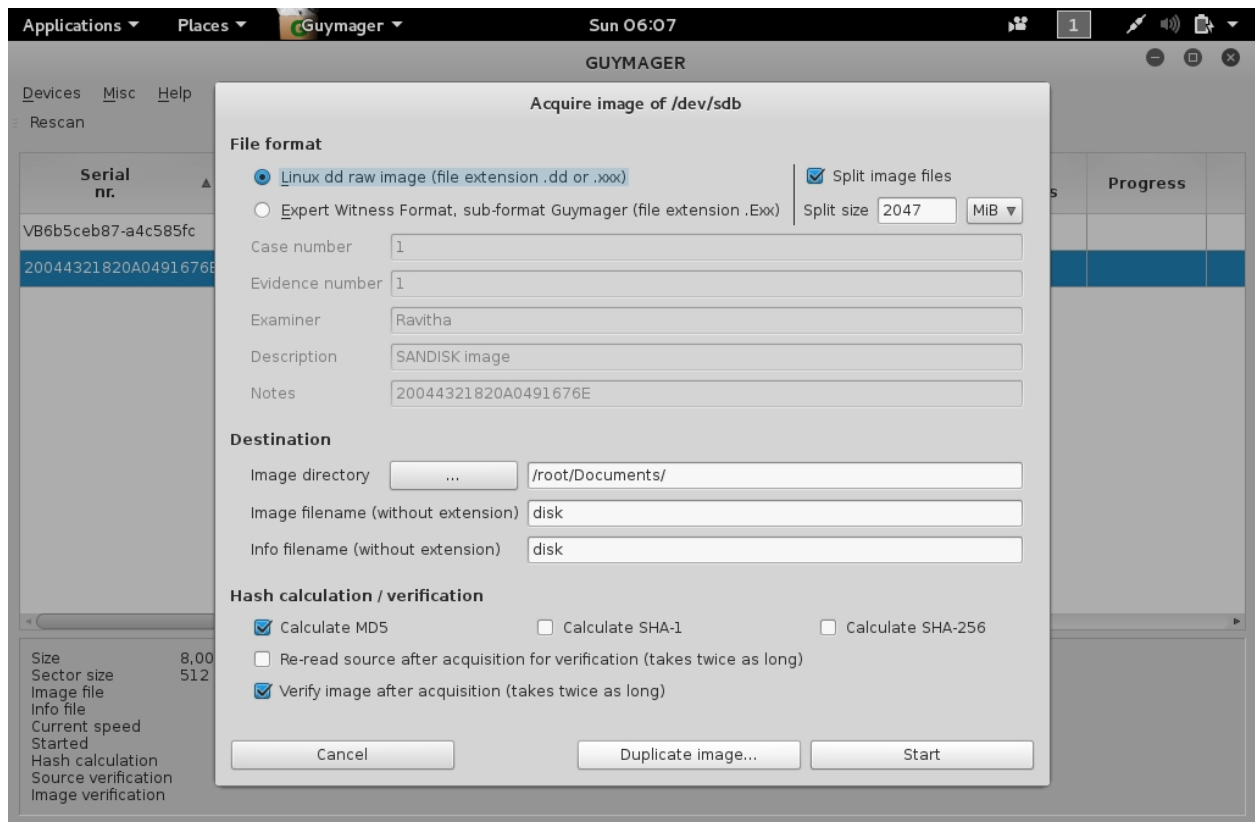
Open Guymager by opening the "Imaging Tools" folder on the desktop and then double clicking on the Guymager icon. When Guymager opens, it will display a list of all mounted disks on the system.



Identify the disk to be imaged, right click on its listing, and select "Acquire image". In this window, first select the disk image format. The options include Linux dd raw image (disk dump utility), Expert Witness Format (.E01). The user has the option to split the image into multiple files, thus making it more easily transferable. So, for example, a 4GB image could be split into four 1GB files, or two 2GB files, etc. After selecting the image format type, fill out the metadata as needed.

Next choose the image directory, which in this example would be "/root/Documents".

Finally, choose the verification options to be performed by Guymager and then Click "Start" to begin the imaging process.



After Guymager has finished creating the disk image, close Guymager and verify the image by navigating to the directory. Notice that there are two files, the image itself and an info file. The info file includes the metadata collected during the acquisition process. The imaging process is now complete.



Report the information generated by guymager by imaging a hard drive with memory less than 1 GB. Split the image files and report extensions used. Can guymager be used for imaging individual folders?

Note: For Imaging Individual Folders in an hard drive, mount the hard drive. Mounting refers to associating the hard drive to the directory tree of linux

Steps to be followed for mounting:

a) Create a directory under home directory

mkdir /mnt/device

b) Mount the device , the device could be present under /dev/sd which could be identified using fdisk
mount /dev/sdb /mnt/device

All the folders in the device could be accessible under the directory /mnt/device

dd

dd is the popular Linux command-line utility for taking images of digital media, such as a hard drive, CD-ROM drive, USB drive, etc., or a particular partition of the media. Images created using dd-like utilities are called raw images, as these are bit-by-bit copies of the source media, without any additions or deletions. The basic usage for dd is:

```
dd if=<media/partition on a media> of=<image_file>
```

Examples include:

```
dd    if=/dev/sdc    of=image.dd  
  
dd    if=/dev/sdc1   of=image1.dd
```

The first example takes the image of the whole disk (sdc), including the boot record and partition table. The second takes the image of a particular partition (sdc1) of the disk (sdc).

Also, instead of storing the image contents in a file, you can pipe the data to a host on the network by using tools such as nc; this is useful when you have space constraints for the image file. For example:

```
dd if=/dev/sdc | nc 192.168.0.2 6000
```

In this example, the contents of the image file are sent via TCP port 6000 to the host 192.168.0.2, where you can run nc in listening mode as nc -l 6000 > image.dd to store the received image contents in the file image.dd.

However, dd has limitations in cases where the media to be imaged has errors in the form of bad sectors.

dd_rescue

dd_rescue is intended specifically for imaging potentially failing media. It switches to a smaller block size (essentially the hardware sector size, most often 512 bytes) when it encounters errors on a drive, and then skips the error-prone sectors, so that it gets as much of the drive as is possible.

dd_rescue will not abort on errors unless one specifies a number of errors (using the -e option) after which to abort. Also, dd_rescue can start from the end of the disk/partition and move backwards. Finally, with dd_rescue, you can optionally write a log file, which can provide useful information about the location of bad sectors. The basic usage of dd_rescue is shown below:

```
dd_rescue <input file> <out file>
```

For example:

```
dd_rescue /dev/sdc image.ddrescue
```



Image a folder in a pen drive by mounting it and by using dd_rescue.

dcfldd

dcfldd is an enhanced version of dd with the following useful additional features for forensics:

- On-the-fly hashing of the transmitted data
- Progress updation for the user
- Splitting of output in multiple files
- Simultaneous output to multiple files at the same time
- Wiping the disk with a known pattern
- Image verification for verifying an image against the original source/pattern

Basic usage for dcfldd is dcfldd <options> if=<input media> of=<image_file>, where <input media> specifies the input media and <image_file> specifies the image file, or the prefix for multiple image files.

Some of the useful options are:

- `split=<bytes>` — Specifies the size of each of the output image files.
- `vf=<file>` — Specifies the image file that needs to be verified against the input.
- `splitformat=<text>` — Specifies the format for multiple image files when using splitting.
- `hash=<names>` — Specifies one or more (a comma-separated list) of hash algorithms, such as md5, sha1, etc.
- `hashwindow=<bytes>` — Specifies the number of bytes of input media for calculation of hash.
- `<hash algorithm>log=<file>` — Specifies output files for hash calculations of a particular hash algorithm.
- `conv=<keywords>` — Specifies conversions of input media according to a comma-separated list of keywords, such as noerror (continue after read errors), ucase (change lower to upper case), etc.
- `hashconv=before|after` — Specifies the hash calculation before or after the specified conversion options.
- `bs=<block size>` — Specifies the input and output block size.

For example:

```
dcfldd if=/dev/hdc hash=md5 hashwindow=10G md5log=md5.txt hashconv=after  
bs=512 conv=noerror,sync split=10G splitformat=aa of=image.dd
```

In this example, 10 gigabytes will be read from the input drive and written to the file image.dd.aa. Then it will read the next 10 gigabytes and write it to file image.dd.ab. Also, an MD5 hash of every 10-gigabyte chunk on the input drive will be stored in the file called md5.txt. The block size is set to 512 bytes, and in the event of read errors, dcfldd will write zeros to the output and continue reading.

Affcat

Affcat is a command-line utility that prints the various contents of an AFF image file. Basic usage is `afcat <options> <AFF file name>`. Some of the useful options are:

- `-p <nnn>` — Output data of page number <nnn>.
- `-S <nnn>` — Output data of sector <nnn> (sector size 512 bytes).
- `-l` — Output all segment names.
- `-L` — Output all segment names, arg and segment length.



Download an AFF image from the net and analyze it using Affcat

LIBEWF

Usage :-

```
ewfacquire -d sha1 -t /root/forensics/ewf/thumb1g /dev/sdb
```

Additional sha1 hash calculation for the image file is provided (newer versions support sha256, but BT R3 doesn't have it by default) and the target file is specified with extension '-t'. Once the tool is started, an interactive menu asks for basic information about the case and the evidence

```
^ v x root@bt: ~
File Edit View Terminal Help
root@bt:~# ewfacquire -d sha1 -t /root/forensics/ewf/thumb1g /dev/sdb
ewfacquire 20100119 (libewf 20100119, libuna 20091031, libbfio 20091114, zlib 1.
2.3.3, libcrypto 0.9.8, libuuid)

Media information:
Device type:          Direct access
Bus type:
Removable:           yes
Vendor:              USB 2.0
Model:               Flash Disk
Serial:
Media size:          1.0 GB (1047527424 bytes)

Acquiry parameters required, please provide the necessary input
Case number: 1
Description: test ewf
Evidence number: 1
Examiner name: cf
Notes: for test
Media type (fixed, removable, optical, memory) [removable]:
Media characteristics (logical, physical) [logical]:
Use compression (none, empty-block, fast, best) [none]:
Use EWF file format (ewf, smart, ftk, encase1, encase2, encase3, encase4, encase
5, encase6, linen5, linen6, ewfx) [encase6]:
Start to acquire at offset (0 >= value >= 1047527424) [0]:
The amount of bytes to acquire (0 >= value >= 1047527424) [1047527424]:
Evidence segment file size in bytes (1.0 MiB >= value >= 7.9 EiB) [1.4 GiB]:
The amount of bytes per sector (0 >= value >= 4294967295) [512]:
The amount of sectors to read at once (64, 128, 256, 512, 1024, 2048, 4096, 8192
, 16384, 32768) [64]:
The amount of sectors to be used as error granularity (1 >= value >= 64) [64]:
The amount of retries when a read error occurs (0 >= value >= 255) [2]:
Wipe sectors on read error (mimic EnCase like behavior) (yes, no) [no]: yes
```

Then it will ask for confirmation, and will start to run:


```
^ v x root@bt: ~
File Edit View Terminal Help
The following acquiry parameters were provided:
Image path and filename:      /root/forensics/ewf/thumb1g.E01
Case number:                  1
Description:                  test ewf
Evidence number:              1
Examiner name:                cf
Notes:                        for test
Media type:                   removable disk
Is physical:                  no
Compression used:             none
EWF file format:              EnCase 6
Acquiry start offset:         0
Amount of bytes to acquire:    999 MiB (1047527424 bytes)
Evidence segment file size:    1.4 GiB (1572864000 bytes)
Bytes per sector:             512
Block size:                   64 sectors
Error granularity:            64 sectors
Retries on read error:        2
Wipe sectors on read error:    yes

Continue acquiry with these values (yes, no) [yes]:

Unable to retrieve serial number.
Acquiry started at: Tue Jan 15 00:41:58 2013

This could take a while.

Status: at 0%.
        acquired 32 KiB (32768 bytes) of total 999 MiB (1047527424 bytes).

Status: at 1%.
        acquired 10 MiB (10485760 bytes) of total 999 MiB (1047527424 bytes).
        completion in 1 minute(s) and 39 second(s) with 9.9 MiB/s (10475274 byte
s/second).

Status: at 2%.
        acquired 20 MiB (20971520 bytes) of total 999 MiB (1047527424 bytes).
```

The following command provides various options

```
ewfacquire -d sha1 -t /root/forensics/ewf/thumb1g -c none -D description -e cf -E 1
-g 64 -f encase6 -m removable -N nonotes -o 0 /dev/sdb
```



```
root@bt: ~  
File Edit View Terminal Help  
root@bt:~# ewfacquire -d sha1 -t /root/forensics/ewf/thumb1g -c none -D description -e cf -E 1 -g 64 -f encase6 -m removable -N nonotes -o 0 /dev/sdb  
ewfacquire 20100119 (libewf 20100119, libuna 20091031, libbfio 20091114, zlib 1.2.3.3, libcrypto 0.9.8, libuuid)  
  
Media information:  
Device type:          Direct access  
Bus type:  
Removable:           yes  
Vendor:              USB 2.0  
Model:               Flash Disk  
Serial:  
Media size:          1.0 GB (1047527424 bytes)  
  
Acquiry parameters required, please provide the necessary input  
Case number: 1  
Media characteristics (logical, physical) [logical]:  
The amount of bytes to acquire (0 >= value >= 1047527424) [1047527424]:  
Evidence segment file size in bytes (1.0 MiB >= value >= 7.9 EiB) [1.4 GiB]:  
The amount of bytes per sector (0 >= value >= 4294967295) [512]:  
The amount of sectors to be used as error granularity (1 >= value >= 64) [64]:  
The amount of retries when a read error occurs (0 >= value >= 255) [2]:  
Wipe sectors on read error (mimic EnCase like behavior) (yes, no) [no]: yes  
  
The following acquiry parameters were provided:
```

With the ewfverify tool It is possible to verify if the stored is the same as a newly calculated one, this way ensuring that the image is not corrupted.

```
ewfverify -d sha1 /root/forensics/ewf/thumb1g.E01
```

```

root@bt: ~
File Edit View Terminal Help

MD5 hash stored in file:      bb756082554440a1844f3a17f2b6ace9
MD5 hash calculated over data: bb756082554440a1844f3a17f2b6ace9
SHA1 hash stored in file:     157eafb8cd3c0e51c072cf774bf788abaf23cff9
SHA1 hash calculated over data: 157eafb8cd3c0e51c072cf774bf788abaf23cff9

ewfverify: SUCCESS
root@bt:~# ewfverify -d sha1 /root/forensics/ewf/thumb1g.E01

```

With ewfinfo, print the metadata of an ewf file.

`ewfinfo /root/forensics/ewf/thumb1g.E01.`

ewfexport allows us to export the raw data or make another ewf format from the file.




The latest version of libewf contains additional tools, like ewfmount which allows us to mount ewf file. This is useful cause the ewf image file contains a lot of extra data, so it's hard to work directly on the raw file.



Create an disk image in E01 format using ewfacquire and verify the hash created using ewfverify

Final Task :-

Find Popular Imaging tools which works under other operating systems and report the findings in the following format.

Tool	Platform			Input Sources				Encoding		Output Formats			
				Physical Disk	Logical Volume	Files	Folders	Compression	Encryption	Raw	E01	Ex01	Split

References

https://en.wikipedia.org/wiki/Live_CD#Mounting_without_burning

http://wiki.bitcurator.net/index.php?title=Creating_a_Disk_Image_Using_Guymager

Gupta, A. , <http://opensourceforu.evytimes.com/2011/03/digital-forensic-analysis-using-backtrack-part-1/>

Vandeven, S. (2014). Forensic Images: for your viewing pleasure. SANS Institute , 1-38.