

CSEC 730 - Advanced Computer Forensics
Sumayyah Fahad Alahmadi
Homework 1

Due Date: Please submit your answers to the Homework 1 dropbox by Sunday midnight 9/16/2018.

This homework comprises three parts.

Part 1. Disk imaging using FTK Imager

Software

Download Windows software FTK Imager Lite from <http://accessdata.com/product-download/ftk-imager-lite-version-3.1.1>

USB Drive

For this activity, you will a USB drive with at least three files of any format on it. You can use any size drive, but using one that is one GB or less will keep the imaging process from being too long. To make this activity realistic, do not use a brand-new drive. Instead, use one that you have loaded and deleted files from over time.

Goal

We have learned that bitstream copies make a bit-for-bit copy of all sectors on a drive. In this activity, you will use a well-known forensics imaging tool, FTK Imager, to create a bitstream image of your USB drive and examine the results.

NOTE: FTK Imager does not guarantee that data is not written to the drive during imaging. For this reason, investigators will use a write blocker when using FTK Imager in a real case. To complete this activity, you can assume that you have a USB write blocker.

A. Create the USB image

Insert your USB to the Windows and launch FTK imager.

Following the following steps, create an image of your USB drive in Raw (dd) format and save the copy to your desktop.

- Select File -> Create Disk Image...
- Choose Physical Drive
- Choose your USB Device
- Press Finish.
- Add the image destination.
- Select Raw (dd) as format.
- Provide destination folder and image filename information.
- Press Start

Task 1. Complete the following multiple choice questions by highlighting the best answers.

1. After the imaging process was complete, what files did FTK Imager create? (Select all that apply.)
 - The image file with extension of .001
 - A text file for image summary
 - An image file with extension of .Ex01
 - No files
2. During the imaging process, you should have noticed that "Verify images after they are created" is checked by default. What is the result to have this option checked? (choose a single best answer)
 - FTK imager will compute the hash value of the image
 - FTK imager will compute the hash value of the USB drive
 - FTK imager will compute the MD5 and SHA1 hashes of the USB drive and the MD5 and SHA1 hashes of the image, and verify the hashes match.
 - FTK imager will compute the MD5 hash of the USB drive and the MD5 hash of the image, and verify the hashes match
3. How many hash algorithms did FTK imager use to verify the image has not been altered? (choose a single best answer)
 - One hash algorithm
 - Two hash algorithms (MD5 & SHA1)
 - Three hash algorithms
 - Four hash algorithms

B. View the acquired USB image

In this exercise, you will load your USB image to FTK imager, and examine the content.

File -> Add Evidence Item...

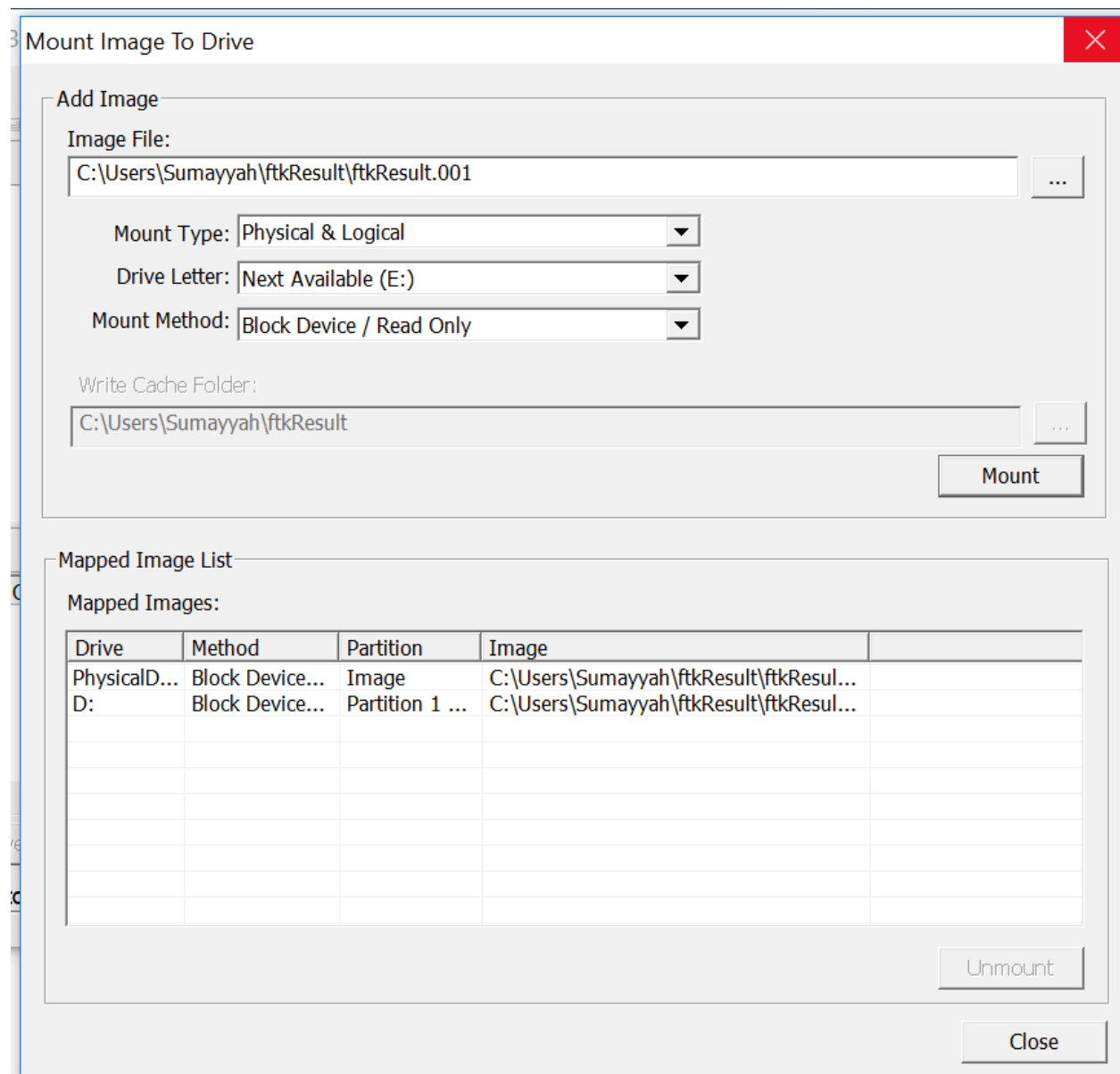
Under *Select the Source Evidence Type*, choose Image File (since we created an image in part A).

USB 1



Task 3. Explore the features supported in FTK Imager, discuss one FTK Imager feature and include a screenshot.

As FTK Imager contains many significant features that support the acquiring process, Image Mount and Unmount consider one of the best features that FTK offer. Mounting is basically mean make the disk image accessible by the computer. In another words, it makes it possible to be read/write by the block device or file system in the computer. Unmounting, in contrast, free up memory space that was located to the mounted image. FTK is not only support moun/unmout, but also offers the choice of making the mounting either physical or logical. And the difference between physical and logical mounting of the image is that the physical mounting copy bit by bit while ignoring EOF indication. logical image mounting, on the other hand, it acquires or copy the content of the disk as it appear in the file system. Moreover, there are three mount methods read only by the block device, read only by the file system or writable by the block device.



Part 2. Imaging with dd and netcat

SANS SIFT Virtual Workstation

SANS SIFT Workstation (virtual machine) has a collection of forensic tools installed. We will use SIFT Workstation for our Linux based homework and labs in this course.

Windows and Linux users can download **VMware Workstation Player**, <https://www.vmware.com/products/player/playerpro-evaluation.html>, a free desktop application that lets you run a virtual machine on a Windows or Linux PC.

Download **SANS SIFT Workstation** at <https://digital-forensics.sans.org/community/downloads>. **You have to create an account** to download the free SANS SIFT Workstation file SIFT-Workstation.ova.

Start the VMware Workstation Player, and select “Player>File>Open”. Navigate to the SIFT-Workstation.ova file and click “Open”. Import the SIFT Virtual machine to your desired location by click “Import”.

After a successful import, you can start the SIFT Workstation. You will be prompted for a username and password for SIFT:

- Default username: *sansforensics*
- Default password: *forensics*

Goal

Sometimes, investigators will capture data from a suspect machine and send data to another networked computer (a forensic machine). *dd* and netcat (*nc*) can be used for imaging over the network. In this activity, you will mimic this process by sending a full image of your own USB from one terminal to another terminal on the same machine using *dd* and *nc*.

Instructions

1. Launch SIFT Workstation 3.
2. Insert **the same USB drive you used in Part 1** and connect the USB to SIFT Workstation 3. The USB drive should auto-mount.
3. Open two terminals on SIFT Workstation 3. One terminal represents a forensic machine; the other represents the suspect machine.
4. On the forensic machine terminal, use `nc -l` to listen on port 8888 for the incoming data. Save the received data as `usb.dd`.
5. On the suspect machine terminal
 - First, generate both MD5 (`md5sum`) and SHA1 (`shasum`) hashes of your USB device
 - Second, use `dd` to make a full image of your USB and pipe (`|`) to netcat (`nc`), sending the USB image to the forensic machine terminal.

Hint: When generating hashes and/or using `dd` command to make a full image of your USB, you need to know the USB device file name as an input. You can run the command `mount` to learn the device file name assigned to your USB device.

For example, my USB's device file name is `/dev/sdc1`

```
/dev/sdc1 on /media/sansforensics/B86D-C764 type vfat (rw,nosuid,nodev,uid=1000,gid=1000,shortname=mixed,dmask=0077,utf8=1,showexec,flush,uhelper=udisks2)
```

6. On the receiving terminal, once you have received the whole USB image, generate both MD5 (`md5sum`) and SHA1 (`shasum`) hashes of your USB image, `usb.dd`
7. Make sure that:
 - The md5 hash of the USB flash matches with the md5 hash of the USB image
 - The sha1 hash of the USB flash matches with the sha1 hash of the USB image.

Task 4: Answer following questions. Please enter your answers carefully, including all spaces and pipes as you would when entering commands to a system.

- What `nc` command did you use on the forensic machine to receive data on port 8888 and save the received data as `usb.dd`?
- What command did you use on the suspect machine to use `dd` to make a full image of your USB and use netcat (`nc`) to send the USB image to the forensic machine terminal?
- What command did you use to generate both MD5 (`md5sum`) and SHA1 (`shasum`) hashes of your USB device in step 5? Include a screenshot.

```
nc -l 8888 > usb.dd
```

```
dd if=sdb1 | nc 172.17.0.1 8888
```

```
sudo md5sum sdb1
```

```
sudo shasum sdb1
```

```
sansforensics@siftworkstation -> /dev
$ sudo md5sum sdb1
33e5591c195b9cd985a28418d7616f5a76 sdb1
```

```
sansforensics@siftworkstation -> /dev
$ sudo shasum sdb1
d3b1900b400384d1d87bb1ea17dd767d76479289 sdb1
```

- What command did you use to generate both MD5 (*md5sum*) and SHA1 (*shasum*) hashes of your USB image in step 6? Include a screenshot.

`sudo md5sum usb2.dd`

`sudo shasum usb2.dd`



```
sansforensics@siftworkstation -> ~ SIFT-REMnux-
$ ncftl8888e> usb2:ddet.pdf Volatility-C
sansforensics@siftworkstation f-> ~ Windows-Fore
$ sudo md5sum usb2.dder.pdf Windows-to-L
33e5591c195b9cd985a28418d7616f5a usb2.dd
sansforensics@siftworkstation -> ~/Desktop
$ sudo shasum usb2.dd
d3b1900b400384d1d87bb1ea17dd767d76479289y usb2.dd
```

- Compare the hash values of *usb.dd* with the hash values of the raw image created by FTK imager in Part 1, are the hash values same or different? Provide a screenshot of the hashes values from the FTK imager to support your answer.

The hash values of the usb is the same as the hash values of the image in SIFT workstation. However, for some reasons these hashes in the SIFT workstation are different than the hashes generated by FTK which is shown in the screenshot below.

Image Verification Results:

Verification started: Sun Sep 16 10:17:03 2018

Verification finished: Sun Sep 16 10:17:34 2018

MD5 checksum: 5ad18a962e4554f127a25fc821dc278b : verified

SHA1 checksum: 8368cc8df27104883f5a341beb207ada8d1e3b5a : verified

Part 3. Linux memory acquisition using LiME

This lab uses SIFT Workstation 3.

Software and environment setting

Download LiME-master.zip from <https://github.com/504ensiclabs/lime>, and extract the zip file.

Goal

Using LiME to dump out the live memory, and try to extract information from the dump.

Part 3.A Using LiME to dump SIFT Workstation's memory

cd to the LiME *src* directory and compile LiME source code using *make*.

Now you should have the kernel module, *lime-VERSION-generic.ko*. Let's load the kernel module and dump out the memory called *yourusername_memory_dump.bin* to your desktop:

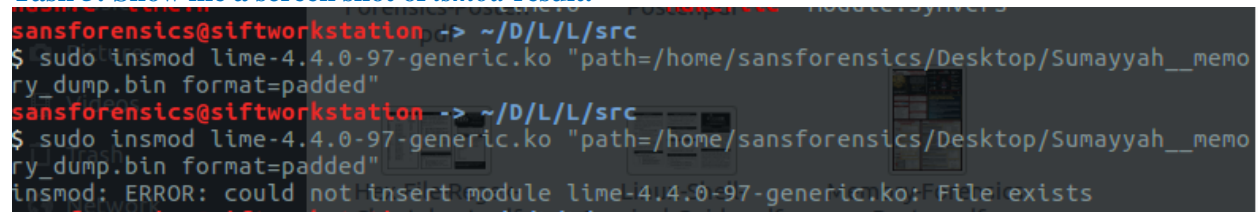
```
insmod lime-VERSION-generic.ko "path=/home/sansforensics/Desktop/yourusername_memory_dump.bin  
format=padded"
```

Note: If you have permission issue to run the *insmod* command: *sudo insmod* ...

After the command is executed, you should see the file, *yourusername_memory_dump.bin* is saved on the desktop.

To check the inserted lime kernel module: *lsmod | grep lime*. You should see "lime" has been inserted.

Task 5: Show me a screen shot of *lsmod* result.



```
sansforensics@siftworkstation -> ~/D/L/L/src  
$ sudo insmod lime-4.4.0-97-generic.ko "path=/home/sansforensics/Desktop/Sumayyah__memo  
ry_dump.bin format=padded"  
sansforensics@siftworkstation -> ~/D/L/L/src  
$ sudo insmod lime-4.4.0-97-generic.ko "path=/home/sansforensics/Desktop/Sumayyah__memo  
ry_dump.bin format=padded"  
insmod: ERROR: could not insert module lime-4.4.0-97-generic.ko: File exists
```



```
sansforensics@siftworkstation -> ~/Desktop
$ ls Music
cases
DFIR-Smartphone-Forensics-Poster.pdf
DFIR-Threat-Intel-Poster.pdf
Find-Evil.pdf
Hex-File-Regex-Cheatsheet.pdf
LIME-master
Linux-Shell-Survival-Guide.pdf
Memory-Forensics-Poster.pdf
mount_points
sansforensics@siftworkstation -> ~/Desktop
$ lsmod | grep lime
grep: command not found
sansforensics@siftworkstation -> ~/Desktop
$ lsmod | grep lime
lime
16384 0
```

To remove the lime kernel module: `sudo rmmod lime`.

Part 3.B Extracting information from your memory dump

Examine the `yourusername_memory_dump.bin` file to find some printable information, such as your login password. For example, you may use `strings` to display printable information (*man strings* to learn how to use this command) and pipe the strings result to `grep` to filter your desired result.

In addition, you may use data carving tools such as *foremost* (installed in SIFT) to extract files based on file headers (Note: using *foremost* is not required for this homework).

Task 6: Show me the command(s) and options you used to extract useful information.

```
sansforensics@siftworkstation -> ~/Desktop
$ cat Sumayyah_memory_dump.bin | strings | grep "forensics"
sansforensics_0
sansforensics
USER=sansforensics
XAUTHORITY=/home/sansforensics/.Xauthority
PWD=/home/sansforensics
HOME=/home/sansforensics
PATH=/home/sansforensics/bin:/home/sansforensics/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
GPG_AGENT_INFO=/home/sansforensics/.gnupg/S.gpg-agent:0:1
LOGNAME=sansforensics
```

Task 7: What type of interesting data you find from your memory?

Memory dump is a great resource to find out juicy information about the user such as password and account name. However, the memory dump contains a lot of information in binary form and in unreadable form so we used tool like `strings` and `grep` to extract the information that we need the most and the information that we can easily read. We find the login user name. Also, all pdf file type as well as img file. (Screenshots provided below)

```

sansforensics@siftworkstation -> ~/Desktop
$ cat Sumayyah__memory_dump.bin | strings | grep passwd
gr_passwd
pw_passwd
pwd.struct_passwd
*user:_Ctype_struct_passwd
**user:_Ctype_struct_passwd
passwd_to_SamInfo3
chgpasswd
chgpasswd['
update_passwd
chgpasswd
chgpasswdV
mksmbpasswdRB
passwd
op=tty_set old-enabled=%d new-enabled=%d old-log_passwd=%d new-log_passwd=%d res=%d
grub-mkpasswd-pbkdf2
nopasswdlogin
passwd
Depends: passwd (>= 1:4.0.3-10)
Depends: passwd (>= 1:4.0.3-10)

```

```

sansforensics@siftworkstation -> ~/Desktop
$ cat Sumayyah__memory_dump.bin | strings | grep pdf
pdf
diffpdf
claws-mail-pdf-viewer
Tpdf
nrpdf
pdf
pdf
MESSAGE=AVC apparmor="STATUS" operation="profile_load" profile="unconfined" name="/usr/
lib/cups/backend/cups-pdf" pid=671 comm="apparmor_parser"
_AUDIT_FIELD_NAME=/usr/lib/cups/backend/cups-pdf
MESSAGE=AVC apparmor="STATUS" operation="profile_replace" profile="unconfined" name="/u
sr/lib/cups/backend/cups-pdf" pid=710 comm="apparmor_parser"
DFIR-Smartphone-Forensics-Poster.pdf
DFIR-Threat-Intel-Poster.pdf
Find-Evil.pdf
Hex-File-Regex-Cheatsheet.pdf
LIME-master
Linux-Shell-Survival-Guide.pdf
Memory-Forensics-Poster.pdf
/etc/fonts/conf.d/99pdf
/usr/share/applications/kde4/okularApplication.pdf.desktop
kde4-okularApplication.pdf.desktop
ion/pdf
xpdf.desktop
/usr/share/applications/xpdf.desktop

```

```

sansforensics@siftworkstation -> ~/Desktop
$ cat Sumayyah_memory_dump.bin | strings | grep img
kernel/drivers/gpu/drm/amd/amdgpu/amdgpu.ko: kernel/drivers/gpu/drm/ttm/ttm.ko kernel/
drivers/gpu/drm/drm_kms_helper.ko kernel/drivers/gpu/drm/drm.ko kernel/drivers/i2c/algo
s/i2c-algo-bit.ko kernel/drivers/video/fbdev/core/fb_sys_fops.ko kernel/drivers/video/f
bdev/core/syscopyarea.ko kernel/drivers/video/fbdev/core/sysfillrect.ko kernel/drivers/
video/fbdev/core/sysimgblt.ko
Xkernel/drivers/video/fbdev/arcfb.ko: kernel/drivers/video/fbdev/core/fb_sys_fops.ko ke
rnel/drivers/video/fbdev/core/syscopyarea.ko kernel/drivers/video/fbdev/core/sysfillrec
t.ko kernel/drivers/video/fbdev/core/sysimgblt.ko
Pkernel/drivers/gpu/drm/ast/ast.ko: kernel/drivers/gpu/drm/ttm/ttm.ko kernel/drivers/gp
u/drm/drm_kms_helper.ko kernel/drivers/gpu/drm/drm.ko kernel/drivers/i2c/algos/i2c-algo
-bit.ko kernel/drivers/video/fbdev/core/fb_sys_fops.ko kernel/drivers/video/fbdev/core/
syscopyarea.ko kernel/drivers/video/fbdev/core/sysfillrect.ko kernel/drivers/video/fbde
v/core/sysimgblt.ko
kernel/drivers/video/fbdev/auc_k1900fb.ko: kernel/drivers/video/fbdev/auc_k190x.ko kern
el/drivers/video/fbdev/core/fb_sys_fops.ko kernel/drivers/video/fbdev/core/syscopyarea.
ko kernel/drivers/video/fbdev/core/sysfillrect.ko kernel/drivers/video/fbdev/core/sysim
gblt.ko
kernel/drivers/video/fbdev/auc_k1901fb.ko: kernel/drivers/video/fbdev/auc_k190x.ko kern
el/drivers/video/fbdev/core/fb_sys_fops.ko kernel/drivers/video/fbdev/core/syscopyarea.
ko kernel/drivers/video/fbdev/core/sysfillrect.ko kernel/drivers/video/fbdev/core/sysim
gblt.ko
kernel/drivers/video/fbdev/auc_k190x.ko: kernel/drivers/video/fbdev/core/fb_sys_fops.ko
kernel/drivers/video/fbdev/core/syscopyarea.ko kernel/drivers/video/fbdev/core/sysfill

```