CSEC 730 ADVANCED FORENSICS

Name : Shriram Karpoora Sundara Pandian (KP)

Title : Homework 1

Content :

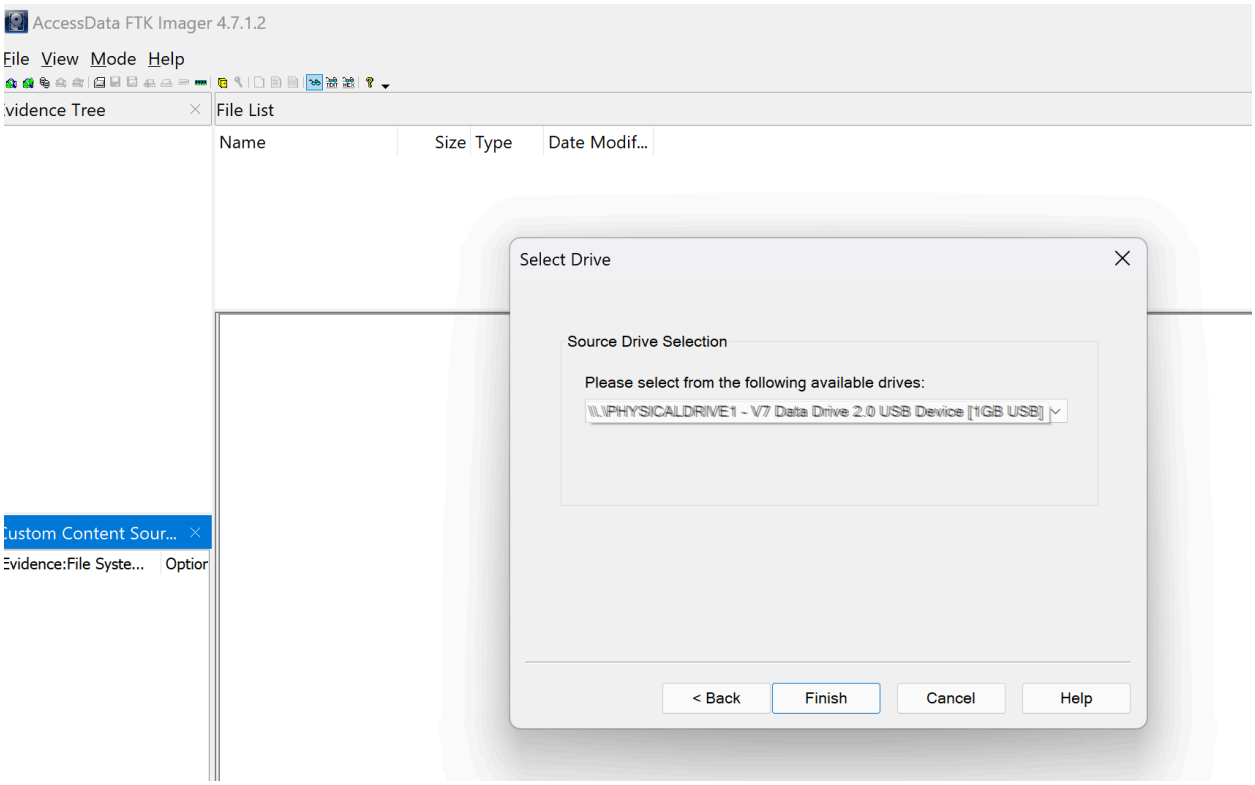
Part 1 : Disk Imaging with FTK Imager

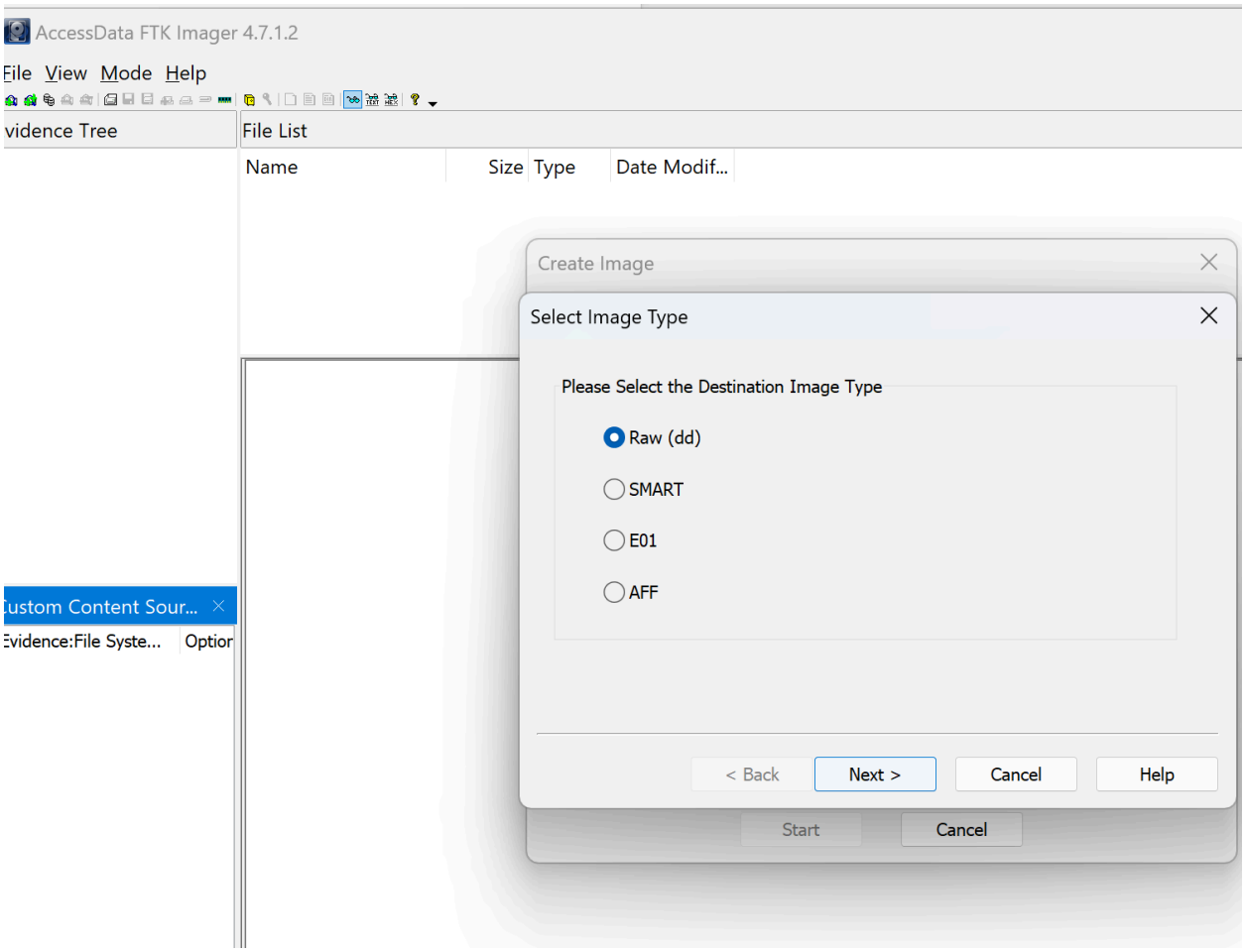
Part 2 : Imaging with DD and Netcat

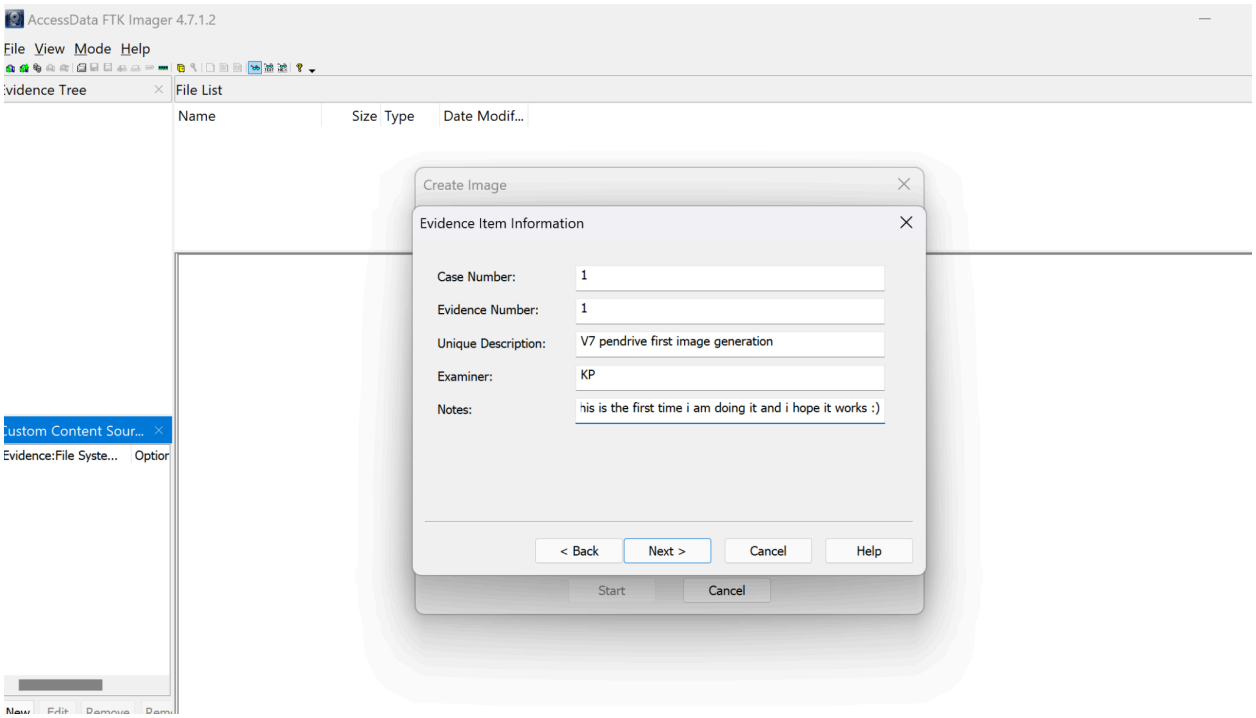
Part 3 : Linux Memory Acquisition using LiME

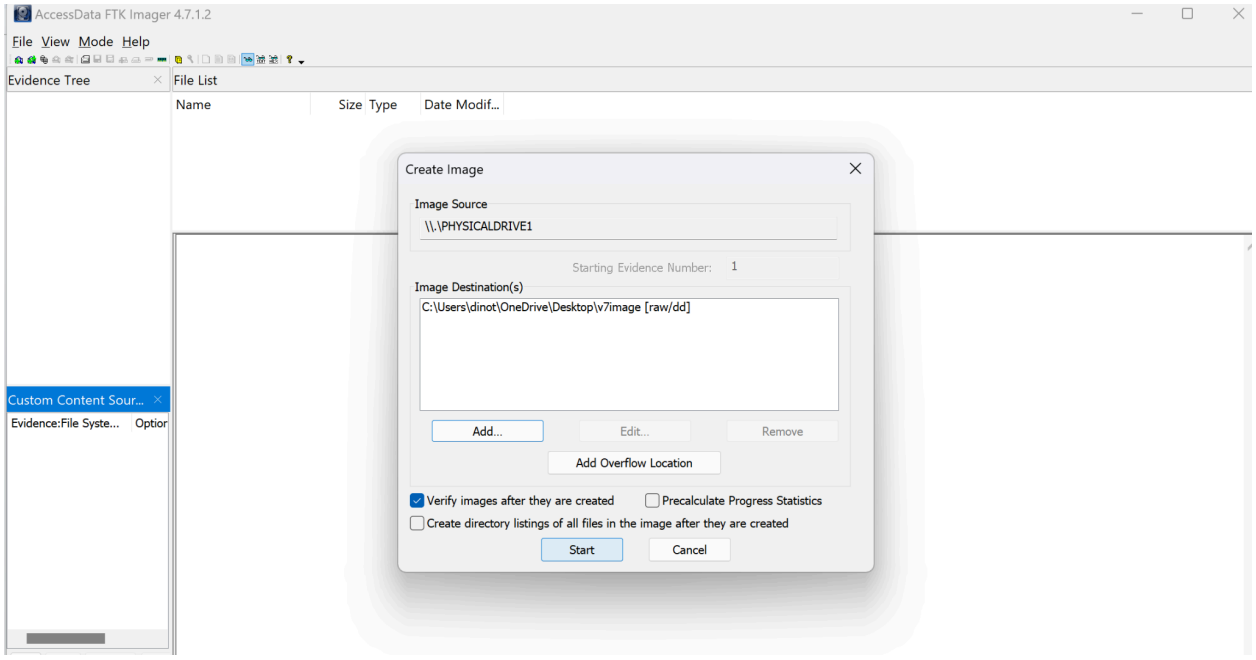
Part 1 : Disk Imaging using FTK imager

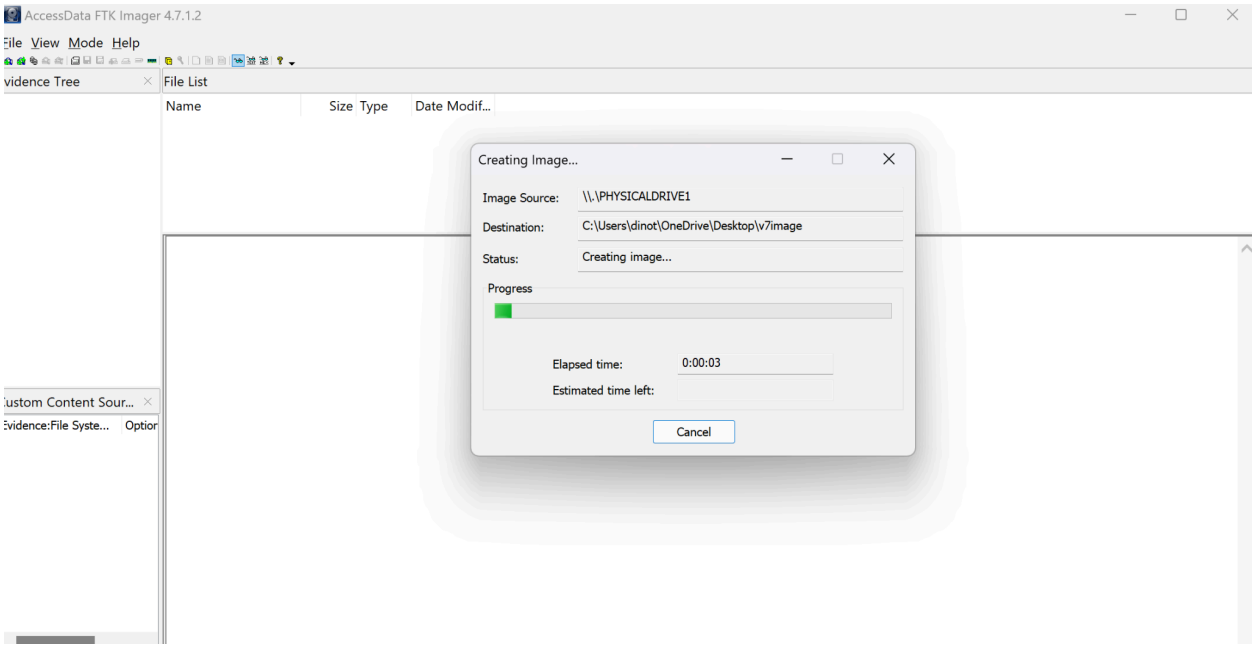
A.

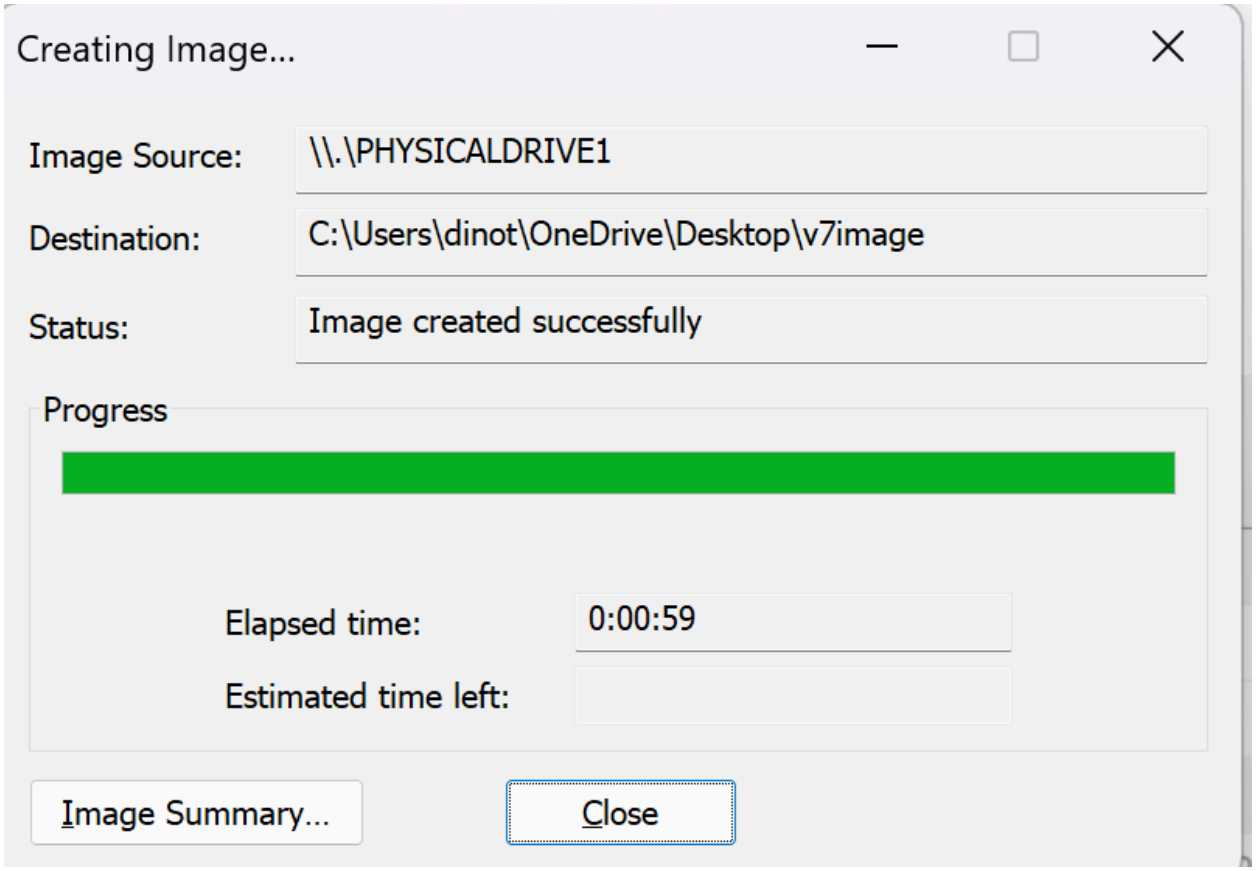



AccessData FTK Imager 4.7.1.2
File View Mode Help
vidence Tree
File List
Name
Size
Type
Date Modif...
Create Image
Select Image Type
Please Select the Destination Image Type
Raw (dd)
SMART
E01
AFF
< Back
Next >
Cancel
Help
Start
Cancel
ustom Content Sour...
Evidence:File Syste...
Optior


AccessData FTK Imager 4.7.1.2
File View Mode Help
File List
Name
Size Type
Date Modif...
Create Image
Evidence Item Information
Case Number: 1
Evidence Number: 1
Unique Description: V7 pendrive first image generation
Examiner: KP
Notes: his is the first time i am doing it and i hope it works :)
< Back
Next >
Cancel
Help
Start
Cancel


AccessData FTK Imager 4.7.1.2
File View Mode Help
Evidence Tree
File List
Name
Size Type
Date Modif...
Create Image
Image Source
\\.\PHYSICALDRIVE1
Starting Evidence Number: 1
Image Destination(s)
C:\Users\dinot\OneDrive\Desktop\v7image [raw/dd]
Add...
Edit...
Remove
Add Overflow Location
Verify images after they are created
Precalculate Progress Statistics
Create directory listings of all files in the image after they are created
Start
Cancel

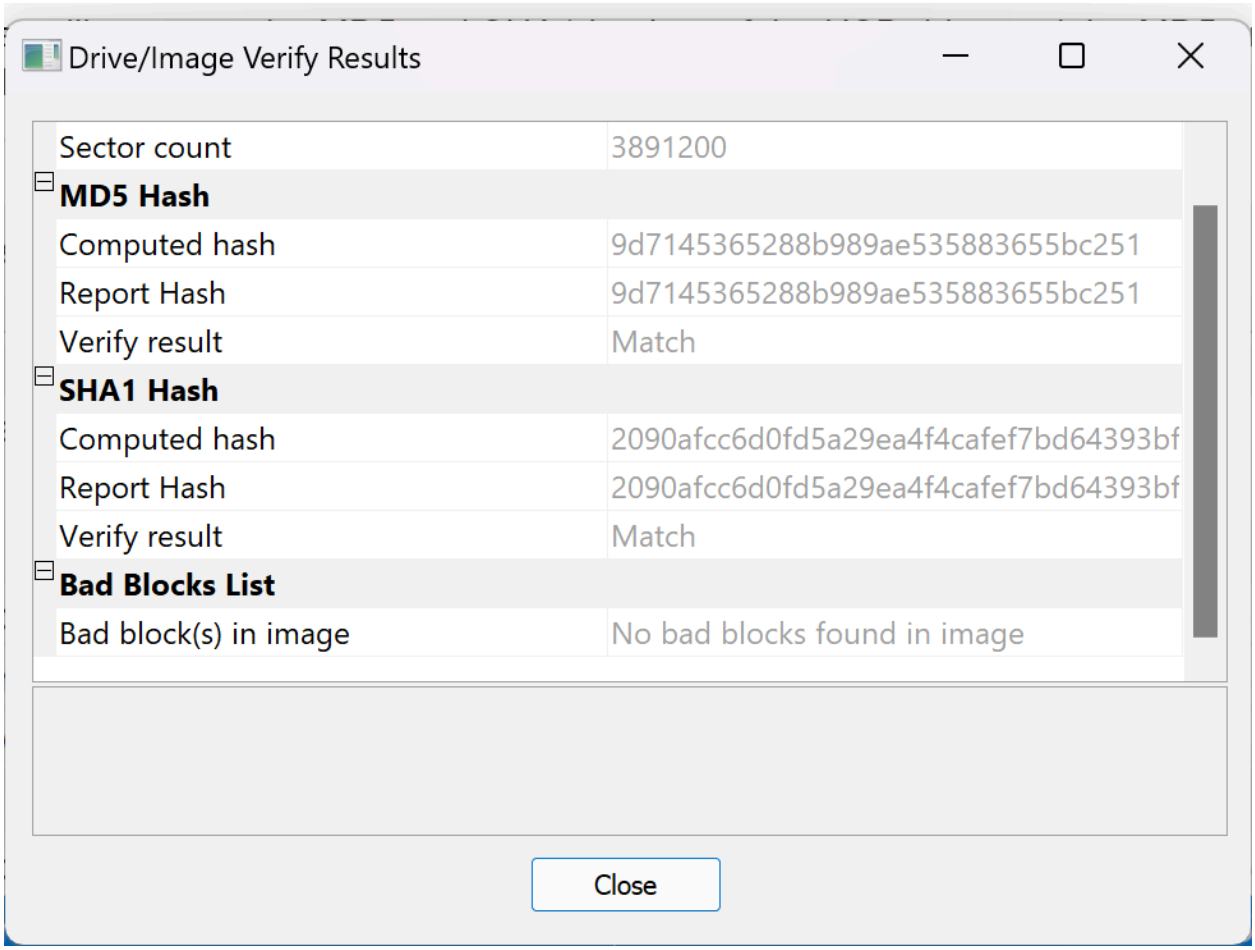

AccessData FTK Imager 4.7.1.2
File View Mode Help
vidence Tree
File List
Name
Size
Type
Date Modif...
Creating Image...
Image Source:
\\.\PHYSICALDRIVE1
Destination:
C:\Users\dinot\OneDrive\Desktop\v7image
Status:
Creating image...
Progress
Elapsed time:
0:00:03
Estimated time left:
Cancel
ustom Content Sour...
vidence:File Syste...
Optior


Creating Image...
Image Source:
\\.\PHYSICALDRIVE1
Destination:
C:\Users\dinot\OneDrive\Desktop\v7image
Status:
Image created successfully
Progress
Elapsed time:
0:00:59
Estimated time left:
Image Summary...
Close

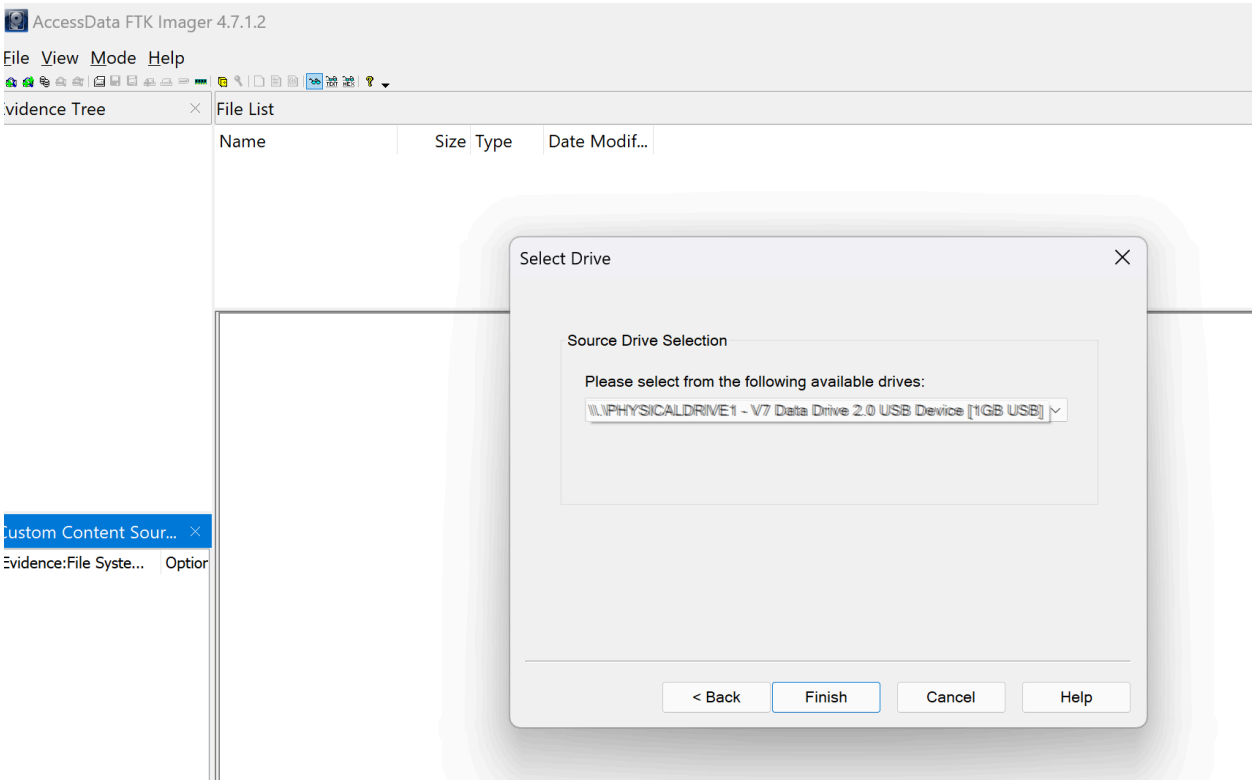
Task 1 :

1. After the images process it created the image file with extension .001

2. 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.
3. Two hash algorithms (MD5 and SHA1)



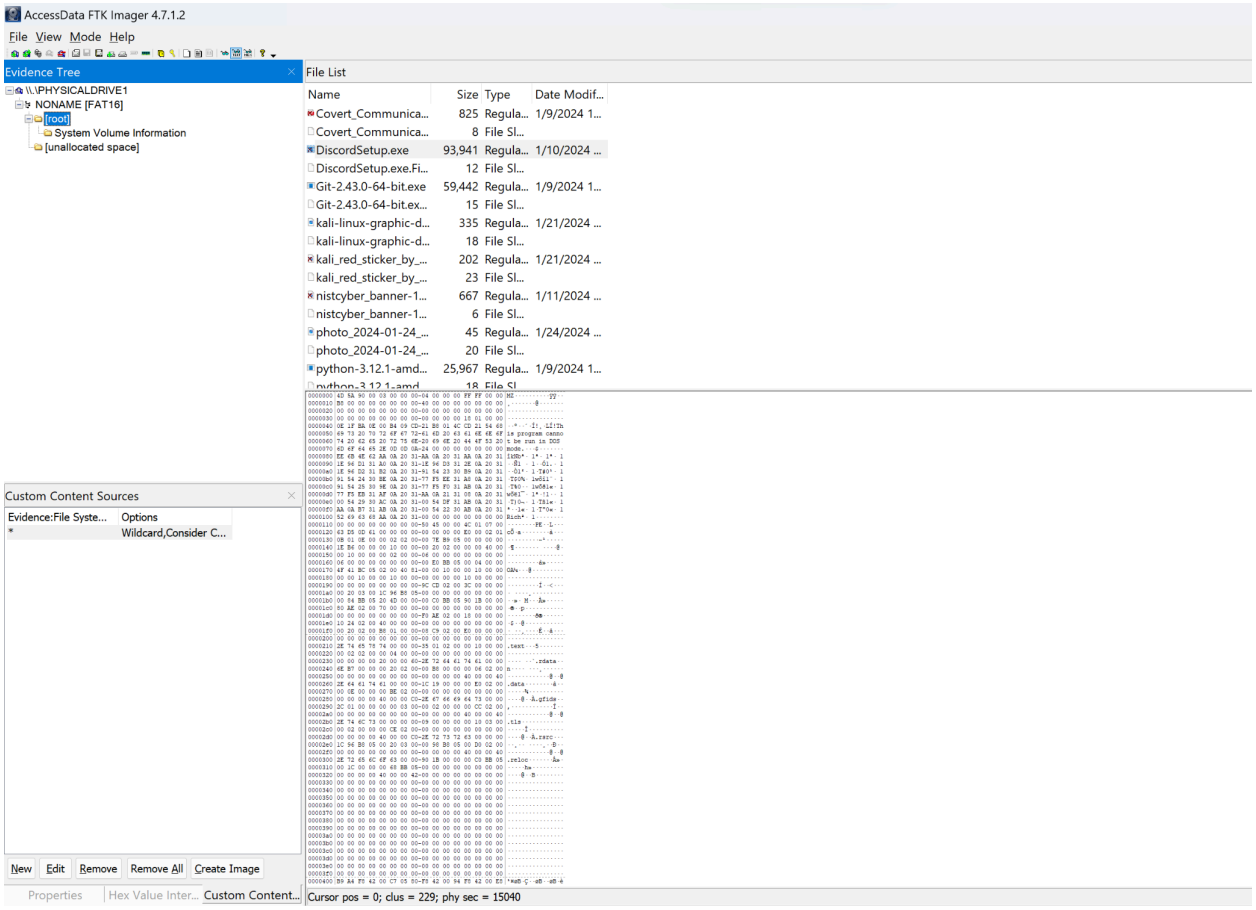
Task 2 :


AccessData FTK Imager 4.7.1.2
File View Mode Help
vidence Tree
File List
Name
Size
Type
Date Modif...
Select Drive
Source Drive Selection
Please select from the following available drives:
\\.\PHYSICALDRIVE1 - V7 Data Drive 2.0 USB Device [1GB USB]
Custom Content Sour...
Evidence:File Syste...
Optior
< Back
Finish
Cancel
Help

This is how the image of the usb image generated look like :

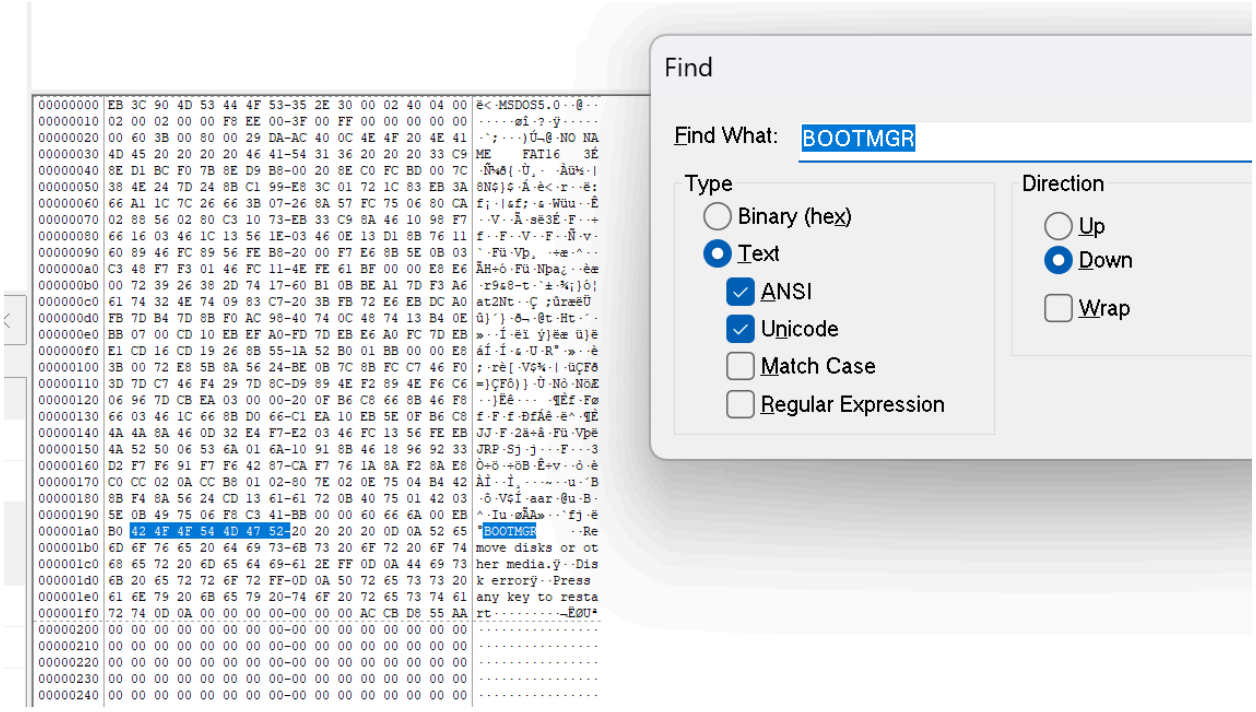
```
Created By AccessData® FTK® Imager 4.7.1.2

Case Information:
Acquired using: ADI4.7.1.2
Case Number: 1
Evidence Number: 1
Unique description: V7 pendrive first image generation
Examiner: KP
Notes: This is the first time i am doing it and i hope it works :)

--------------------------------------------------------------

Information for C:\Users\dinot\OneDrive\Desktop\v7image:

Physical Evidentiary Item (Source) Information:
[Device Info]
 Source Type: Physical
[Drive Geometry]
 Cylinders: 242
 Tracks per Cylinder: 255
 Sectors per Track: 63
 Bytes per Sector: 512
 Sector Count: 3,891,200
[Physical Drive Information]
 Drive Model: V7 Data Drive 2.0 USB Device
 Drive Serial Number: 2312061912143525975003
 Drive Interface Type: USB
 Removable drive: True
 Source data size: 1900 MB
 Sector count:    3891200
[Computed Hashes]
 MD5 checksum:    18a6d842578adc726644c21a4982e383
 SHA1 checksum:   8b6b55c9ff599ad049d89b99ffd040581306a815

Image Information:
 Acquisition started:   Thu Jan 25 17:59:29 2024
 Acquisition finished:  Thu Jan 25 18:00:29 2024
 Segment list:
  C:\Users\dinot\OneDrive\Desktop\v7image.001
  C:\Users\dinot\OneDrive\Desktop\v7image.002

Image Verification Results:
 Verification started:  Thu Jan 25 18:00:30 2024
 Verification finished: Thu Jan 25 18:00:39 2024
 MD5 checksum:    18a6d842578adc726644c21a4982e383 : verified
 SHA1 checksum:   8b6b55c9ff599ad049d89b99ffd040581306a815 : verified
```

Task 3 :

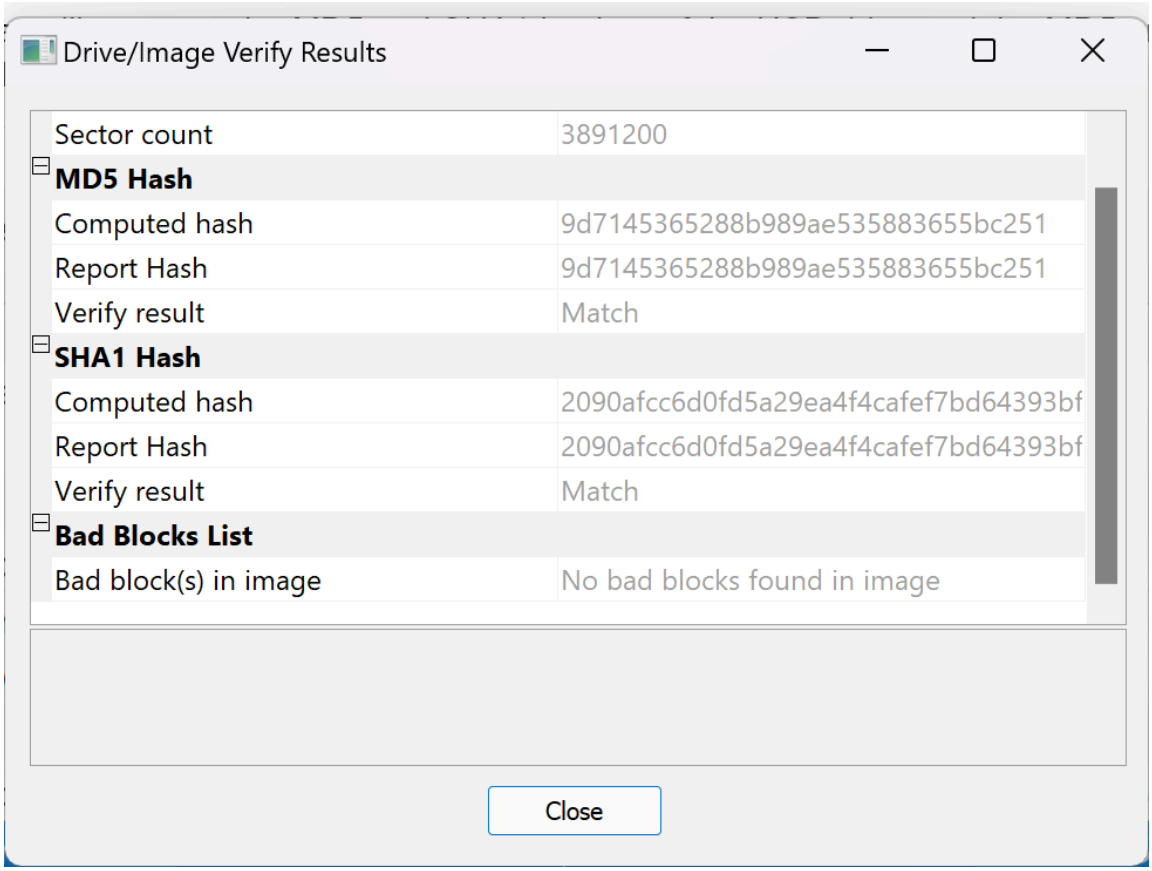


I tried to load my pendrive as > Add an evidence item and opened it further for analysis as FTK imager is powerful tool and I tried to view the hex values of the .exe files because lot of malwares will be in .exe or .bat file in windows. So it really helped to view hex format for listing the evidence and work on it further as a proof.

I can also view a particular text in the hex view of the .exe file for verification or guessing

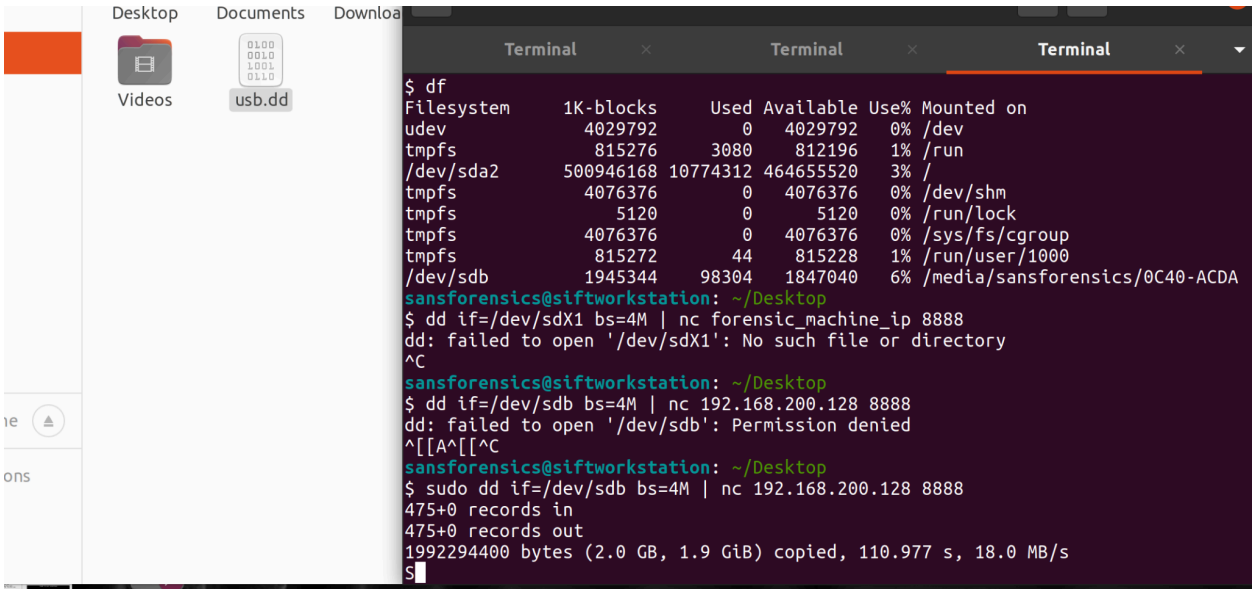


We can also verify the pendrive before opening it which is very useful



These are further options we can play around our evidence which is V7 Pendrive in my case.

Part 2 : Imaging with dd and netcat (nc)



Task 4 :

1. · What *nc* command did you use on the forensic machine to receive data on port 8888 and save the received data as *usd.dd*?

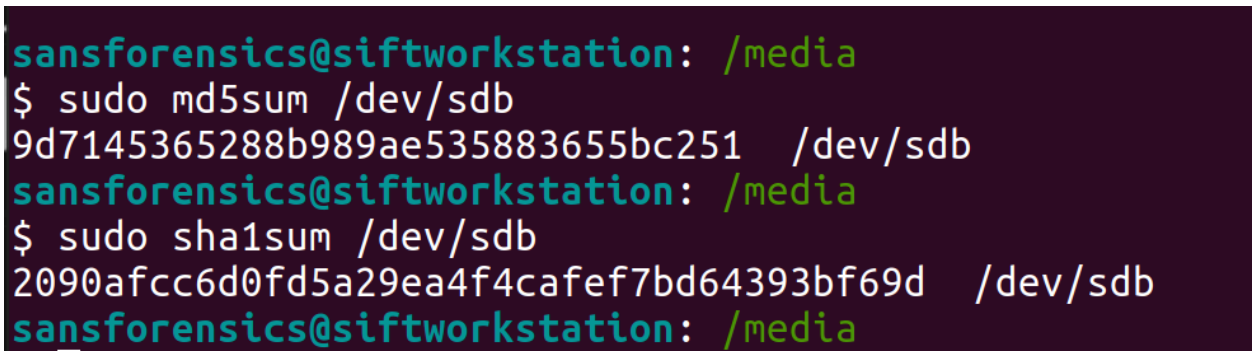
```
sansforensics@siftworkstation: ~
$ nc -l 8888 > usb.dd
sansforensics@siftworkstation: ~
$ ls
Desktop    Downloads    md5hash.txt  md5usb.txt  Pictures  sha1hash.txt  usb.dd
```

2. · 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?

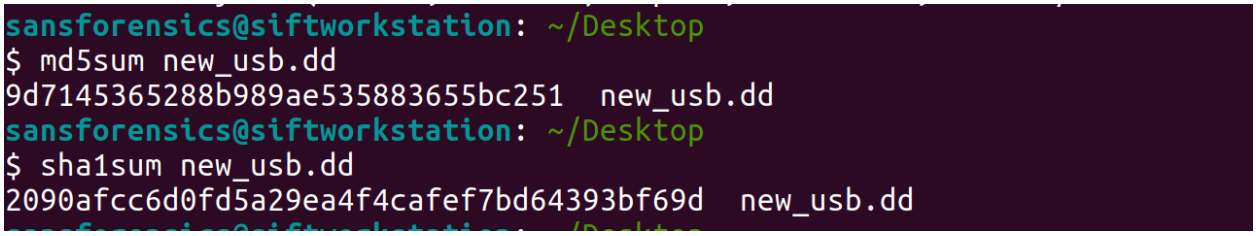
   My USB device name is sdb in my case.

```
sansforensics@siftworkstation: ~/Desktop
$ sudo dd if=/dev/sdb of=usb2.dd | nc 192.168.200.128 8888
3891200+0 records in
3891200+0 records out
1992294400 bytes (2.0 GB, 1.9 GiB) copied, 79.6365 s, 25.0 MB/s
```

3. · What command did you use to generate both MD5 (*md5sum*) and SHA1 (*shasum*) hashes of your USB device in step 5? Include a screenshot.



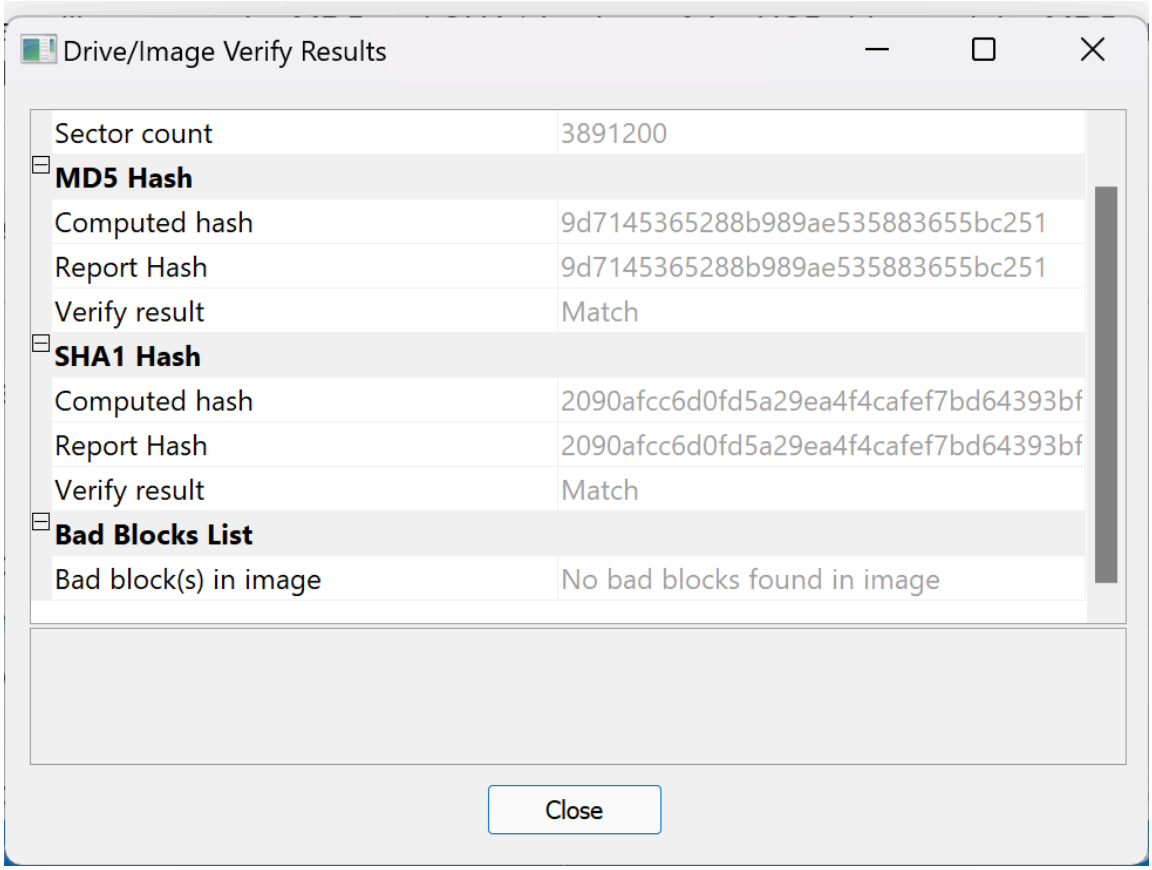
4. · What command did you use to generate both MD5 (*md5sum*) and SHA1 (*shasum*) hashes of your USB image in step 6? Include a screenshot.



From the above two images from 3 and 4 hashes of device and its image is matching perfectly.

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

```
sansforensics@siftworkstation: ~/Desktop
$ md5sum new_usb.dd
9d7145365288b989ae535883655bc251  new_usb.dd
sansforensics@siftworkstation: ~/Desktop
$ sha1sum new_usb.dd
2090afcc6d0fd5a29ea4f4cafef7bd64393bf69d  new_usb.dd
```



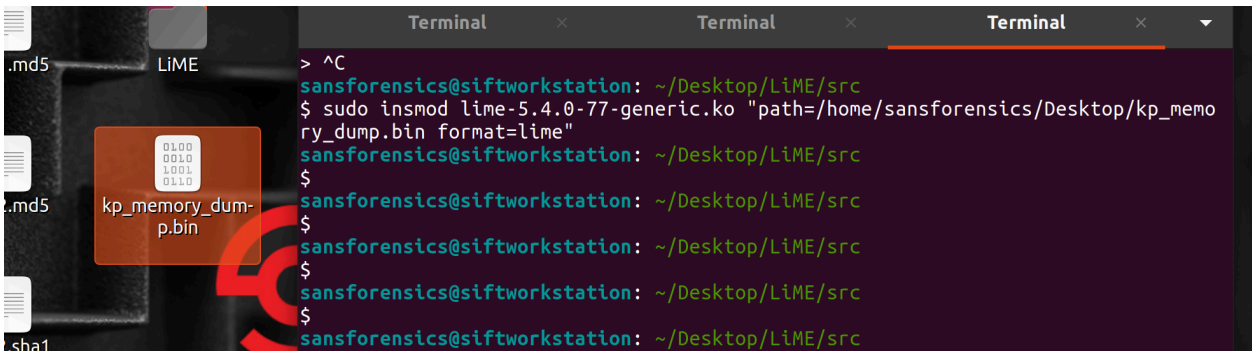
Both the hashes from FTK imager and from SIFT VM (MD5 and SHA1) are same.. Because both bit by bit stream copy (duplicate) of the original file and both ftk and dd uses the same method to clone it, hence the file in not modified and just cloned, the hashes are same still. But I feel this result is odd though because I didn't use writeup and results keep changing on different times.

Part 3 : Linux memory Acquisition

So we are going to acquire the volatile data from the live linux system by dumping using the LiME tool.

```
sansforensics@siftworkstation: ~/Desktop
$  git clone https://github.com/504ensicsLabs/LiME.git
Cloning into 'LiME'...
remote: Enumerating objects: 370, done.
remote: Counting objects: 100% (21/21), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 370 (delta 10), reused 12 (delta 4), pack-reused 349
Receiving objects: 100% (370/370), 1.61 MiB | 5.83 MiB/s, done.
Resolving deltas: 100% (199/199), done.
```

```
sansforensics@siftworkstation: ~/Desktop/LiME/src
$ make
make -C /lib/modules/5.4.0-77-generic/build M="/home/sansforensics/Desktop/LiME/
src" modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-77-generic'
  CC [M]  /home/sansforensics/Desktop/LiME/src/tcp.o
/home/sansforensics/Desktop/LiME/src/tcp.c: In function ‘setup_tcp’:
/home/sansforensics/Desktop/LiME/src/tcp.c:75:5: warning: ISO C90 forbids mixed
declarations and code [-Wdeclaration-after-statement]
   75 |     int opt = 1;
      |     ^~~
  CC [M]  /home/sansforensics/Desktop/LiME/src/disk.o
  CC [M]  /home/sansforensics/Desktop/LiME/src/main.o
  CC [M]  /home/sansforensics/Desktop/LiME/src/hash.o
  CC [M]  /home/sansforensics/Desktop/LiME/src/deflate.o
  LD [M]  /home/sansforensics/Desktop/LiME/src/lime.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/sansforensics/Desktop/LiME/src/lime.mod.o
  LD [M]  /home/sansforensics/Desktop/LiME/src/lime.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-77-generic'
strip --strip-unneeded lime.ko
mv lime.ko lime-5.4.0-77-generic.ko
```

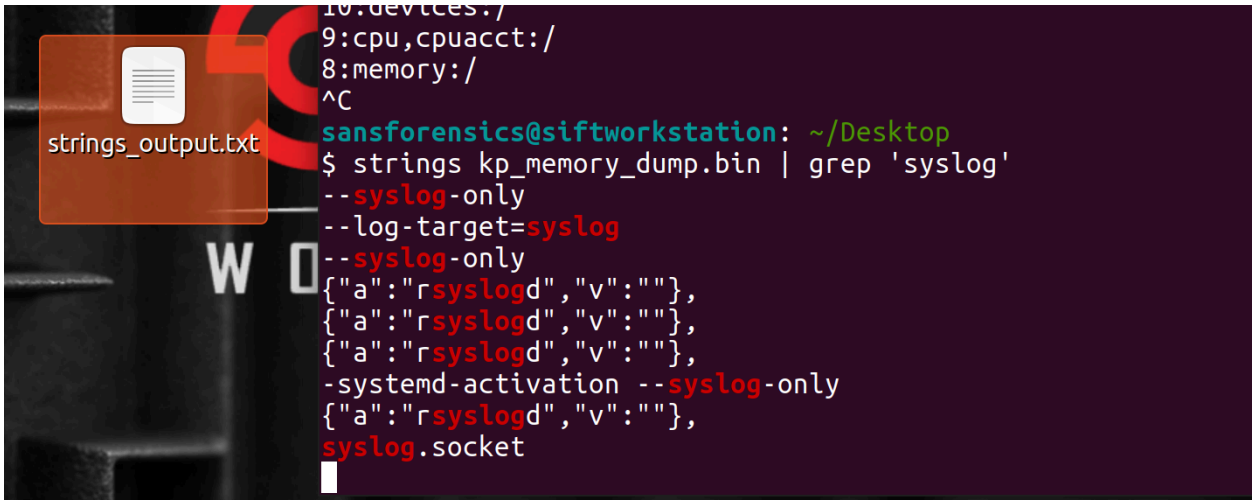


Task 5

```
sansforensics@siftworkstation: ~/Desktop/LiME/src
$ lsmod | grep lime
lime                   16384  0
sansforensics@siftworkstation: ~/Desktop/LiME/src
```

Task 6 :

From below screenshots I am able to retrieve the syslogs, Ip address and email address present in dump file and able to do lot of things with strings command.

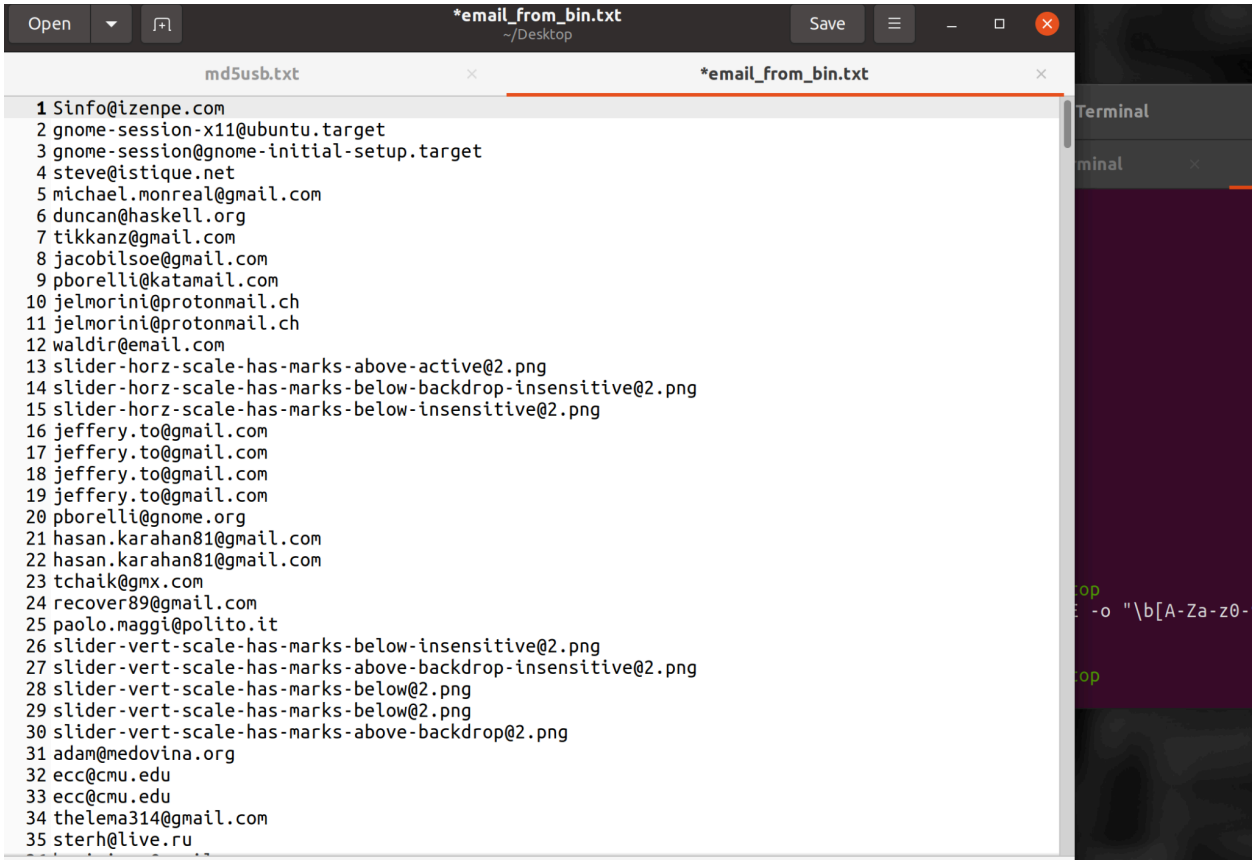



```
StandardOutput=syslog
rsyslog.conf
rsyslog.d
mc_set_syslog
__vsyslog_chk
syslog
__vsyslog_chk
__syslog_chk
rsyslogd
__syslog_chk
__syslog_chk
pam_syslog
pam_vsyslog
rsyslog.conf
rsyslog.d
__vsyslog_chk
log_set_upgrade_syslog_to_journal
__vsyslog_chk
../src/basic/syslog-util.c
syslog_parse_priority
syslog-or-kmsg
cap_syslog
```

```
-systemd-activation --syslog-only
enesyslogic.gl3590.firmware
ww.genesyslogic.com/en/product_list.php?1st=3
syslog.service
#define __NR_syslog 103
#define __NR_ia32_syslog 103
syslog:/pN>
tw.com.genesyslogic.gl3590.firmware
http://www.genesyslogic.com/en/product_list.php?1st=3
syslog.service
/lib/systemd/system/rsyslog.service
/usr/sbin/rsyslogd
man:rsyslogd(8)
https://www.rsyslog.com/doc/
/usr/sbin/rsyslogd
```

To extract email address from the bin file :

```
$ strings kp_memory_dump.bin | grep -E -o "\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A
-Za-z]{2,6}\b"
info@izenpe.com
gnome-session-x11@ubuntu.target
gnome-session@gnome-initial-setup.target
steve@istique.net
michael.monreal@gmail.com
duncan@haskell.org
tikkanz@gmail.com
jacobilsoe@gmail.com
pborelli@katamail.com
jelmorini@protonmail.ch
jelmorini@protonmail.ch
waldir@email.com
slider-horz-scale-has-marks-above-active@2.png
slider-horz-scale-has-marks-below-backdrop-insensitive@2.png
slider-horz-scale-has-marks-below-insensitive@2.png
jeffery.to@gmail.com
jeffery.to@gmail.com
jeffery.to@gmail.com
jeffery.to@gmail.com
pborelli@gnome.org
hasan.karahan81@gmail.com
```

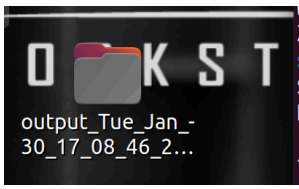



```
sansforensics@siftworkstation: ~/Desktop
$ strings kp_memory_dump.bin | grep -E -o "(25[0-5]|2[0-5][0-5]|[01]?[0-9][0-9]?)
\.(25[0-5]|2[0-5][0-5]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-5][0-5]|[01]?[0-9][0-9]?)\
.(25[0-5]|2[0-5][0-5]|[01]?[0-9][0-9]?)" > ip_from_bin.txt
sansforensics@siftworkstation: ~/Desktop
$ cat ip_from_bin.txt | head -10
192.168.200.2
192.168.200.128
192.168.200.254
1.222.71.48
1.222.71.48
0.0.0.0
4.1.29.1
1.222.71.48
192.168.200.2
255.255.255.0
sansforensics@siftworkstation: ~/Desktop
```

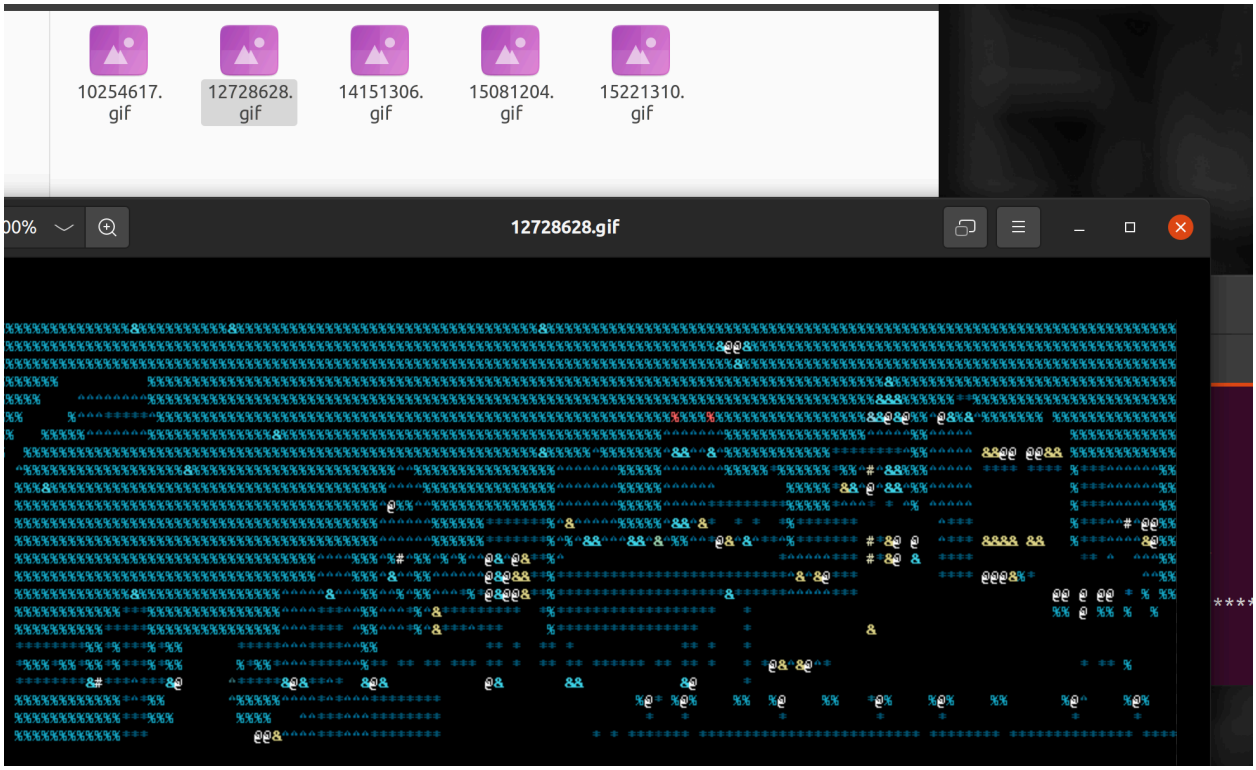
Task 7 :

Foremost is shockingly nice tool and retrieved jif and pdf files and as we can see the pdf file it has extracted has a research paper I have done for my last semester and it's a shock for me to extract it using foremost.

```
sansforensics@siftworkstation: ~/Desktop
$ foremost -v -i kp_memory_dump.bin -o foremost-output
Foremost version 1.5.7 by Jesse Kornblum, Kris Kendall, and Nick Mikus
Audit File

Foremost started at Mon Jan 29 22:53:59 2024
Invocation: foremost -v -i kp_memory_dump.bin -o foremost-output
Output directory: /home/sansforensics/Desktop/foremost-output
Configuration file: /etc/foremost.conf
Processing: kp_memory_dump.bin
|------------------------------------------------------------------
File: kp_memory_dump.bin
Start: Mon Jan 29 22:53:59 2024
Length: 7 GB (8589404288 bytes)

Num      Name (bs=512)         Size      File Offset     Comment

*****************************0:        06325876.exe         781 KB      3238848640
*************1: 08828549.exe         781 KB      4520217137
2:      09010780.exe         209 KB      4613519488
*3:     09065901.doc          11 MB      4641741728
*4:     09278396.exe         781 KB      4750538880
**5:    09749137.exe         781 KB      4991558321
***6:   10299690.exe         781 KB      5273441712
******************7:    14089333.doc          11 MB      7213738992
******8:        15200804.doc          11 MB      7782811784
****9:  16003866.doc          11 MB      8193979424
10:     16086156.exe         781 KB      8236112000
****|
Finish: Mon Jan 29 22:55:35 2024

11 FILES EXTRACTED

doc:= 4
exe:= 7
------------------------------------------------------------------
```




```
Windows-to-Unix-Cheatsheet.pdf
Zimmerman-Tools-Poster.pdf
sansforensics@siftworkstation: ~/Desktop
$ foremost -i kp_memory_dump.bin -t gif -T
Processing: kp_memory_dump.bin
|*******************************************************************
**********|
```




```
sansforensics@siftworkstation: ~/Desktop
$ foremost -i kp_memory_dump.bin -t pdf -T
Processing: kp_memory_dump.bin
|*******************************************************************************
***********|
```

A Covert Channel Using Public USB Charging Stations

1st Natan Tapper
Department of Cybersecurity
Rochester Institute of Technology
Rochester, New York
nst1340@rit.edu

2nd Shriram Karpoora Sundara Pandian
Department of Cybersecurity
Rochester Institute of Technology
Rochester, New York
sk2410@rit.edu

3rd Daryl Johnson
Department of Cybersecurity
Rochester Institute of Technology
Rochester, New York
Daryl.Johnson@rit.edu

***Abstract*—Public spaces like airports, cafes, and malls present challenges for covert messaging due to open visibility and lack of control over infrastructure. However, they also provide access to shared resources that could enable hidden communication channels. This paper investigates the feasibility of a covert timing channel utilizing common public USB charging stations. We propose modulating the electrical current from a charging port to encode binary data that is passively observed by a mobile device when connected. An Arduino-based transmitter is implemented to control the charging current by varying resistance. A custom mobile app monitors changes in battery charge current to decode the binary signal. We also present a novel periodic table encoding scheme to further obfuscate messages. While unable to fully implement the channel, our proof-of-concept demonstrates the potential for practical covert messaging hiding in plain sight. With further work, this approach may provide usable, flexible covert communication tailored for diverse public environments. The study lays an exploratory foundation for covert timing channels leveraging ubiquitous public infrastructure.**

***Index Terms*—Covert Channel, Communication, Charging, Android**

I. Introduction

The term "covert channel" was first coined in 1973 by Butler Lampson in his paper "A Note on the Confinement Problem." [1] In 1993, Virgil Gligor defined covert channels as "a communication channel that allows a process to transfer information in a manner that violates the system's security policy." [2] Covert communication channels allow the transfer of information in a manner that aims to avoid detection. Covert channels can use various techniques such as steganography to hide messages in media, while other methods leverage indirect signalling by modulating shared resources [3]. Timing channels in particular encode information in the timing of events and do not rely on direct data transmission.

Public spaces like airports, cafes, and malls present challenges for covert messaging due to a lack of control over infrastructure and open visibility. However, such environments also enable access to shared resources that could facilitate hidden communication. Public spaces allow for hiding a covert channel in plain sight. If done well, it can create a very effective covert channel that no one will suspect or recognize as a communication channel. For example, USB charging stations in public areas could offer a conduit for establishing covert timing channels using power or electromagnetic side channels. This paper examines the feasibility of creating a covert timing channel based on public charging stations.

Public charging stations have become widely adopted in public spaces. Now more than ever, everyone carries electronic devices that need to be charged. It is often viewed as a basic service to provide access to charging stations in areas such as airports, cafes, malls, or hotels.

In this paper, we investigate a covert channel utilizing public charging stations. The data is sent by varying the current offered by the charger. An increase in current is used to indicate a binary 1 while a decrease in current is used to indicate a binary 0. Through this method, we can design a covert channel that can largely avoid detection while hiding in plain sight.

The goal of this research is to develop a proof-of-concept covert timing channel using a common public charging station as a conduit. The system should offer simple deployment, realistic throughput for text messaging, and resilience against common disruption methods. This paper presents implementation details, results, and analysis of the proposed public charging station covert channel. We also present a novel encoding scheme that utilizes the periodic table to avoid the covert message becoming known should the covert channel be compromised.

To our knowledge, this is the first work using a phone charger to send covert messages to a phone. Other works have looked at covert channels using the phone as the sending however not from using the charger as the sender messaging the phone.

The rest of the paper is laid out as follows: Section 2 provides some background information and definitions that are crucial for understanding the remainder of the paper. Section 3 details related work and similar papers in the space. Section 4 details the design of our covert channel. Section 5 is a discussion of the results from our testing. Section 6 is a brief conclusion and Section 7 details future areas of research surrounding this topic.

II. Background

In this section, we will discuss concepts that are crucial for the understanding of this paper.